Parameteroptimierung des Ballmodells humanoider Fußball spielender Roboter

Oscar Salvador Morillo Victoria
o_morillo@yahoo.de

Bachelorarbeit
Freie Univesität Berlin
Fachbereich Mathematik und Informatik

Betreuer
Prof. Dr. Raúl Rojas
Dr. Hamid Reza Moballegh

Berlin, 10.02.2014

Zusammenfassung

Mit Hilfe dynamischer Systeme werden in der Robotik Objekte der realen Welt modelliert. Diese beinhalten Parameter, die unmittelbaren Einfluss auf die Güte der Modelle haben. In der Domäne von humanoiden Fußball spielenden Robotern werden damit u.a. Ball und Hindernisse, wie beispielsweise das Tor und gegnerische Spieler und nichtzuletzt auch die Selbstlokalisierung des Roboters umgesetzt.
Für die Modelle des Team FUmanoid wurden die Parameter bislang manuell justiert, was einen zeitaufwändigen Prozess darstellt. In dieser Arbeit wird am Beispiel des auf einem erweiterten Kalman Filter basierenden Ballmodells gezeigt, dass die Parameteroptimierung solcher Modelle automatisierbar ist und auch bessere Ergebnisse liefert.
Zur Optimierung wurde ein Optimierungsframework implementiert, welches Partikelschwarmoptimierung (PSO), eine stochastische, populationsbasierte Metaheuristik aus der Kategorie Schwarmintelligenz, verwendet.
Mit Hilfe des Frameworks lassen sich vergleichbar gute Parametersätze wie die manuell justierten in kurzer Zeit, aber auch bessere bei längerer Ausführung bestimmen. Dank der Parallelisierung des Frameworks konnte eine bis zu 30-fache Beschleunigung des Optimierungsprozesses erreicht werden. Das Framework wurde in einer eigens entwickelten Clusterumgebung mit bis zu 11 Knoten und 67 CPU-Kerne getestet.
Zusammenfassend lässt sich sagen, dass PSO erfolgreich auf die Bestimmung von Ballmodel-Parameter angewendet werden kann und wirft die Frage auf, ob eine Optimierung der Hindernismodell- und Selbstlokalisierungsparameter ebenfalls erreicht werden kann.

Inhaltsverzeichnis

1
Einleitung

Objekterkennung und Objektlokalisierung sind zwei wichtige Vorverarbeitungsschritte in der Robotik, so auch bei humanoiden Fußball spielenden Robotern. Um einen Ball ins Tor zu befördern, also mit der Umwelt zu interagieren, müssen diese in der Lage sein, ihre Umgebung zu verstehen - *wo ist ein Ball, wo ist ein Hindernis* und *welches ist unser Tor*. Ein gutes Ball- und Hindernismodell sowie eine gute Selbstlokalisierung sind somit essentiell, um erfolgreich Fußball spielen zu können und tragen neben einer guten Motion und klugen Verhaltensstrategien wesentlich zum Ausgang eines Spieles bei.

Um die Position des Balles während eines Spieles zu schätzen, wird dessen Zustand in ein zeit-invariantes Modell überführt, welches die Zustandsschätzung ermöglicht. Das Modell beeinhaltet Parameter, die es zu bestimmen gilt. Bisher wurden diese Parameter manuell bestimmt, was sehr zeitaufwändig ist. In dieser Arbeit möchte ich am Beispiel des Ballmodells zeigen, dass die Suche nach geeigneten Parametern für solche Modelle automatisierbar ist und sich mit Hilfe eines geeigneten Verfahrens Parametersätze bestimmen lassen, die mit den zeitaufwändig manuell justierten vergleichbar sind.

1.1 Motivation

Um eine Handlung wie beispielsweise das Befördern des Balles ins Tor erfolgreich planen zu können, bedarf es Wissen über den Zustand der Umwelt des Roboters. Kenntnisse über die aktuelle Ballposition und seine Geschwindigkeit, die Roboterpose und Informationen über Hindernisse wie z.B. die gegnerischen Spieler sind dabei von Bedeutung.

Um diese Informationen zu erlangen, greift der Roboter seine Sensordaten - Odometrie und Kamera - ab, welche jedoch meist nicht rauschfrei sind. Die Odometrie ist nie exakt und unterscheidet sich von Roboter zu Roboter, aber auch die Kamera kann falsche Schlussfolgerungen suggerieren, wie z.B. fälschlicherweise ein als Ball erkanntes Objekt.

Mit Hilfe der Sensordaten versucht man, ein physikalisches Modell über die Welt zu erstellen, was allerdings nur möglich ist, wenn der Zustand zeitabhängig modelliert wird, da physikalische Parameter wie z.B. Geschwindig-

keit über die Zeit definiert sind.

Um Odometrie und Kamera zu fusionieren, also in einem Modell des realen Systems zu vereinen, ist ein naiver Ansatz den vorherigen und den aktuellen Zustand zu vergleichen und darüber eine Schätzung über den nächsten Zustand zu inferieren. Dieses Modell ist aufgrund des Sensorrauschens jedoch sehr ungenau. Um die Schätzung besser und stabiler zu machen, wird das Rauschen explizit mitmodelliert und über die Zeit integriert.

S. Otte implementierte in seiner Masterarbeit ein neues Ballmodell sowie ein Hindernismodell und eine Selbstlokalisierung, welche alle Zustandschätzung mit Hilfe des Kalman Filters umsetzen.(1) Das Rauschen ist dabei durch Parameter modelliert, die es zu bestimmen gilt. Je akkurater die Schätzung der Parameter, desto besser funktioniert der Filter.

1.2 Der RoboCup

> *"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup."*
>
> *– http://www.robocup.org/about-robocup/objective/ (2)*

Der RoboCup ist eine Weltmeisterschaft, bei der Wissenschaftler und Studenten ihre Fußball spielenden Roboter gegeneinander antreten lassen. Er findet jährlich an wechselnden Orten statt. Das langfristige Ziel des RoboCups ist es, Mitte des 21. Jahrhunderts eine Mannschaft aus Robotern aufzustellen, welche die Mannschaft, die als Gewinner aus der FIFA Weltmeisterschaft hervorgeht, nach den offiziellen FIFA Regeln besiegt.
Die Arbeitsgruppe *Intelligente Systeme und Robotik* der Freien Universität Berlin unter der Leitung von Prof. Raúl Rojas González nahm bereits seit 1999 mit den FU-Fighters am RoboCup teil. Seit 2008 nimmt eine zweite Mannschaft der Arbeitsgruppe, die *FUmanoids*, jährlich daran teil. In dieser Arbeitsgruppe ist auch diese Bachelorarbeit entstanden.

1.3 Problemstellung

Das lokale[1] Ballmodell entspricht im Wesentlichen dem von S. Otte implementierten Modell, lediglich sind zwei weitere Parameter in der aktuellen Implementierung vorhanden, die im Absatz *Bemerkungen* kurz erläutert werden. Für eine detailliertere Beschreibung des lokalen Ballmodells verweise ich auf seine Masterarbeit (1).

[1] *lokal* deutet daraufhin, dass die Ballposition relativ zum Roboter angeben ist und nicht in Weltkoordinaten.

Das Ballmodell wird durch einen erweiterten Kalman Filter modelliert, der es im Gegensatz zu einem Kalman Filter auch ermöglicht, nicht-lineare Probleme abzubilden.
Der Kalman Filter ist ein Gaussfilter, mit deren Hilfe, rekursive Zustandsschätzung von dynamischen Systemen betrieben werden kann. Dabei wird der Zustand des Systems durch die beiden Momente, Mittelwert μ und der Kovarianzmatrix Σ, repräsentiert. Im Falle des Ballmodells dient dieser da-

zu eine Zustandsschätzung des Balles abzugeben.

Der Zustand des Balles ist dabei durch die beiden Komponenten, seiner Position und seiner Geschwindigkeit, modelliert. Wird der Ball nicht gesehen, so wird mit Hilfe der letzten geschätzten Position des Balles und mit seiner zuletzt prädizierten Geschwindigkeit sowie der Odometrie des Roboters, die neue aktuelle Position des Balles geschätzt. Sobald der Ball wieder sichtbar für den Roboter wird, werden der Zustand des Modells und des tatsächlich gesehenen Balles miteinander abgeglichen und das Ballmodell aktualisiert.

Ein Problem, welches sich bei der Schätzung ergibt und das ein weiteres Arbeitsfeld des Kalman Filters darstellt ist, dass die Daten von den Sensoren, wie beispielsweise der Odometrie des Roboters, mit deren Hilfe die Unsicherheit im Modell des Balles reduziert wird, nicht rauschfrei sind. Um das Rauschen zu filtern bzw. zu minimieren, verwaltet der Kalman Filter sogenannte Rauschmatrizen, die additiv in der Berechnung des neuen Zustands, also der neuen Position und Geschwindigkeit, mit eingehen. Man unterscheidet zwischen den Kovarianzrauschmatrizen P und Q. P beschreibt den Fehler, der bei der Datenerhebung entsteht und Q die Ungenauigkeit, die aufgrund der Vorhersage zum Modell hinzukommt. Je akkurater die Rauschmatrizen P und Q sind, desto besser funktioniert der Filter, was letztlich eine bessere Schätzung der Ballposition ermöglicht.

Die Bestimmung der Rauschmatrizen erfolgte bislang durch exploratives Testen. Parameter zu finden, die das Rauschen gut beschreiben, sind dabei schwer zu finden und ist in der Regel sehr zeitaufwändig.

Tabelle 1.1 zeigt die Parameter der Rauschmatrizen P und Q für das Ballmodell.

Parameter
friction
process_update.invalid_error_x_threshold
process_update.invalid_error_y_threshold
process_update.noise_pos
process_update.noise_vel
sensor_update.noisepitch
sensor_update.noiseyaw

Tabelle 1.1: Rauschmatrizen-Parameter des Ballmodells

Bemerkung Die in Tabelle 1.1 aufgeführten Parametern *process-update-invalid-error-x-threshold* und *process-update-invalid-error-y-threshold* stellen Grenzwerte für die Unsicherheit des Ballmodells dar. Überschreitet die Unsicherheit der x-Koordinate bzw. der y-Koordinate den jeweiligen Grenzwert, so wird das Ballmodell invalidiert.

Problem Das Problem, das es in dieser Arbeit zu lösen gilt, ist die automatische Schätzung der beiden Kovarianzmatrizen P und Q durch ein Optimierungs-Framework.

1.4 Struktur

Die Arbeit gliedert sich wie folgt. Im zweiten Kapitel wird die Wahl für Schwarmintelligenz-Verfahren begründet und Werke mit Bezug vorgestellt sowie die Grundlagen von Schwarmintelligenz erläutert. In Kapitel 3 wird der Lösungsansatz formuliert, der in Kapitel 4 umgesetzt wurde. Das nachfolgende Kapitel beschäftigt sich mit den Experimenten und der Evaluierung der Implementierung. Das letzte Kapitel fasst die Ergebnisse zusammen und diskutiert diese.

2
Theoretischer Hintergrund

In dieser Arbeit geht es um die Optimierung von Parametern. Da keine Eigenschaften der zu optimierenden Zielfunktion bekannt sind, wurden Schwarmintelligenzverfahren zur Optimierung herangezogen. Schwarmintelligenzverfahren sind stochastische, populationsbasierte Optimierungsverfahren. Der stochastische Ansatz führt dazu, dass Regionen im Suchraum auch erkundet werden, wenn sie nicht in der Nähe vom Optimum liegen, was die Möglichkeit bietet, ein lokales Optimum wieder zu verlassen. Des Weiteren zeichnen sich diese Verfahren durch den Austausch von Informationen zwischen den Individuen aus, was eine gezielte Suche nach dem Optimum fördert . Der nachfolgende Abschnitt *Verwandte Werke* stellt einige Arbeiten vor, die sich mit der gleichen Fragestellung beschäftigen. Anschließend wird auf die Grundlagen von Schwarmintelligenz-Verfahren mit Schwerpunkt auf Partikelschwarmoptimierung, eingegangen.

2.1 Verwandte Werke

Jatoth et al. schätzen die Kovarianzmatrizen eines Extended Kalman Filters und zeigen, dass mit ihrem Ansatz bessere Kovarianzmatrizen gefunden werden als die manuell bestimmten (3). Als Gütefunktion zur Bewertung eines Parametersatzes verwenden sie MSE (*mean square error*). Talatahari et al. wenden Ameisenkolonie-Optimierung auf die Parameterschätzung von Fluthäufigkeitsverteilungen an (4). Majhi et al. nutzen eine inkrementelle Variante des PSO Algorithmus um *first order system*-Parameter robust und verteilt zu bestimmen (5). Wang et al. machen Gebrauch von einer chaotischen Version des PSO Algorithmus um die optimalen Parameter und eine optimale Eingabeuntermenge für das *grey model* zu schätzen (6). Zhou et al. optimieren die Parameter des nichtlinearen Bernoulli-Modells mit Hilfe eines Partikelschwarmoptimierers (7). Niehaus et al. verwenden Partikelschwarmoptimierung, um den durch parametrizierbare Trajektorien modellierten Gang eines humanoiden Roboters zu optimieren (8). Mit Hilfe von Partikelschwarmoptimierung gelingt es Burchardt et al. die Partikelfilterparameter für die Selbstlokalisierung von humanoiden Fußball spielende Roboter zu verbessern (9).

2.2 Schwarmintelligenz

> *Definition Schwarmintelligenz: "Any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies"*
>
> *– Bonabeau, Eric (10)*

Neben den explizit angegebenen Arbeiten geben die nachfolgenden Abschnitte den Inhalt der folgenden Quellen sinngemäß wieder: (11), (12) sowie (13).

2.2.1 Kollektive Intelligenz

Kollektive Intelligenz, auch Gruppen- oder Schwarmintelligenz genannt, lässt sich in vielen natürlichen Systemen wie beispielsweise bei Mensch und Tier wiederfinden. Kollektive intelligente Verhaltensweisen bei Tieren lassen sich z.B. bei Vögeln, Fischen, Bakterien, Insekten und Ameisen beobachten. Das kollektive Zusammenspiel der Ameisen ermöglicht es ihnen beispielsweise schwere Beute zu ihrem Nest zu transportieren.

– Karatay,Mehmet (14)

Abbildung 2.1: Bestimmte Verhaltensmuster der Ameisen die als Grundlage für einen Optimierungsalgorithmus diente

Die intelligente Verhaltensweise beruht dabei nicht auf der Verhaltensweise eines Einzelnen, sondern ergibt sich aus der Summe der Verhaltensweisen aller Individuen.
Besonders in der Nahrungssuche wie unter anderem bei Vögeln und im Schutz vor Feinden wie beispielsweise bei Fischen (siehe 2.2) bietet die Schwarmbildung erhebliche Vorteile. Die Gemeinschaft hat mehr Erfolg bei der Suche nach Nahrung als der Einzelne und auch die Chance einem Feind zu entkommen, erhöht sich in der Gemeinschaft (Verwirrung des Feindes).

– Brocken Inaglory (15)

Abbildung 2.2: Raubfische sind durch den Fischschwarm visuell überfordert *predator confusion effect.*

Der Begriff *Schwarmintelligenz* ist von der Künstlichen Intelligenz geprägt

BEGRIFFSHERKUNFT

und fiel erstmalig im Jahre 1988 in der Publikation von Beni et al. im Zusammenhang mit Zellularen Robotik Systemen (16). In der Arbeit ging es um intelligente Agenten, die einfache Aktionen ausführen, um ein größeres Muster zu erzeugen. Die Agenten handeln dabei selbstorganisierend. Die Autoren bezeichneten die Gruppen der intelligenten Agenten als Schwarm, da *"[..] diese Eigenschaften aufweisen, die sonst bei Insektenschwärme aufzufinden [..]"* seien wie *"dezentralisierte Kontrolle, fehlende Synchronization, einfache und quasi identische Mitglieder"*(16).

Die zwei wesentlichen Eigenschaften, welche die Schwarmintelligenz charakterisieren sind dabei **Selbstorganisation** und **Arbeitsteilung**.
Bonabeau erweiterte den Begriff der Schwarmintelligenz, der ursprünglich für Zellulare Robotik Systeme Verwendung fand (siehe 2.2) und formulierte vier Eigenschaften, auf dessen selbstorganisierende Systeme beruhen:

Positives Feedback Durch eine Aktion kann ein Individuum die anderen Individuen der Gemeinschaft in ihrer Handlung beeinflussen. Als Beispiel sei an dieser Stelle das Tanzen der Bienen erwähnt, mit deren Hilfe diese, andere Bienen auf die Richtung zu einer Nektar- oder Pollenquelle aufmerksam machen.

Negatives Feedback Auf der anderen Seite kann ein Feedback auch dazu dienen, um beispielsweise auf eine Gefahr oder auf eine ausgeschöpfte Quelle hinzuweisen. In solch einem Falle bezeichnet Bonabeau dieses als negatives Feedback.

Fluktuation Fluktuationen sind zufällige Verhalten der Individuen um neue Zustände zu erforschen, wie z.B. zufälliges Fliegen von Honigbienen, um neue Nektarquellen zu finden.

Multiple Interaktionen Alle Arten von selbstorganisierenden Systemen beruhen nicht zuletzt auf der Fähigkeit der Individuen auf unterschiedliche Einflüsse zu reagieren und unterschiedlich zu interagieren.

Mittlerweile sind zahlreiche Algorithmen zur Optimierung entstanden, die von unterschiedlichen staatenbildenden Insekten und anderen in Schwärme lebenden Tieren wie Ameisen, Bienen sowie einigen Fisch- und Vogelarten inspiriert sind. Die beliebtesten Algorithmen darunter sind die Partikelschwarmoptimierung sowie die Ameisenkolonie-Optimierung. Diese und weitere Algorithmen aus der Kategorie der Schwarmintelligenz werden seither erfolgreich auf komplexe Probleme der realen Welt angewendet, wie z.B. auf das Traveling Salesman Problem (TSP) (17), auf das Satisfiability Problem (SAT) (18) und auf Non-Linear Programming-Probleme (NLP) (19).
Ein wichtiges Anwendungsfeld von Schwarmintelligenz-Algorithmen ist die Robotik. Das unter dem Begriff *Swarm Robot* zusammengefasste Forschungsfeld kombiniert Schwarmintelligenz mit der Koordination von Multirobotsystemen. Bonabeau nennt als Gründe für diese Richtung der Schwarm

basierten Roboter, die Flexibilität und die Robustheit sowie die dezentrale und selbstorganisierende Variante des Problemlösens durch soziale Insekten. (10)
Die Vorteile von Schwarmintelligenz-Algorithmen liegen laut I. Kassabalidis(20) in der Skalierbarkeit, der Fehlertoleranz[1] , der Anpassungsfähigkeit[2], der Geschwindigkeit, der Modularität, der Autonomie sowie in der Parallelität.

[1] Die Aufgabe wird fertiggestellt, auch wenn einige Individuen fehlschlagen.

[2] Der Schwarm passt sich inneren und äußeren Einflüssen an.

Im Folgenden werden drei ausgewählte Algorithmen der Schwarmintelligenz vorgestellt: **Künstlicher Bienenkolonie-Algorithmus (ABC)**, **Ameisenkolonie Optimierung (ACO)** sowie **Partikelschwarmoptimierung (PSO)**.

2.3 Künstlicher Bienenkolonie-Algorithmus

2.3.1 Honigbienen in der Natur

Bienenschwärme zeigen bei ihren täglichen Aufgaben viele intelligente Verhaltensweisen, u.a. beim Bienenstock bauen, bei der Navigation, bei der Aufgabenselektion und nicht zuletzt bei der Nahrungssuche. Letztere diente auch als Vorbild für den künstlichen Bienenkolonie-Algorithmus (ABC).
Honigbienen werden nach ihrer Rolle bei der Aufgabenverteilung unterschieden. Diese lassen sich in zwei Rollen kategorisieren - angestellte Bienen und nicht angestellte Bienen.

Angestellte Bienen Ist eine Biene angestellt, so transportiert sie Nektar oder Pollen aus einer Pflanze zum Bienenstock. Auf dem Weg dorthin, tritt sie in den sogenannten Tanzbereich ein, in welchem sie die gefundene Pflanze bewirbt. Das Anwerben von neuen Rekruten erfolgt durch den sogenannten Bienentanz. Mit Hilfe von zwei Tanzbewegungen kommunizieren die Bienen die Qualität der Quelle.

POSITIVES FEEDBACK

Nicht angestellte Bienen Nicht angestellte Bienen können sich auf zweierlei Weisen verhalten. Zum einen können sie die im Tanzbereich vollführten Tänze beobachten, um sich für eine gute Nektarquelle zu entscheiden. Zum anderen können sie selbst die Umgebung nach einer neuen Quelle erforschen, wobei sie durch interne oder externe Einflüsse geleitet werden.

MULTIPLE INTERAKTIONEN

FLUKTUATION

2.3.2 Der ABC-Algorithmus

Angelehnt an das Vorbild in der Natur, simuliert der von Karaboga entwickelte Künstliche Bienenkolonie-Algorithmus drei Bienenarten - die angestellte Biene, den Ausspäher und die beobachtende Biene. Die Ausspäher modellieren die nicht-angestellte Biene, welche die Umgebung erforschen und beobachtende Biene, diese, die im Tanzbereich nach guten Quellen Ausschau halten.
Die Bienenkolonie setzt sich zu Beginn zur Hälfte aus angestellten Bienen und zur anderen Hälfte aus beobachtenden Bienen zusammen. Jeder angestellten Biene ist eine Futterquelle zugeordnet, sobald die Futterquelle einer angestellten Biene erschöpft ist, wird diese zu einem Ausspäher.
Der Pseudocode nach (12) ist in Algorithmus 1 dargestellt.

Algorithm 1 ABC - Pseudocode

1: sende die Ausspäher los zu den initialen Futterquellen
2: **while** Abbruchbedingung nicht erfüllt **do**
3: Sende die angestellten Bienen zu den Futterquellen und ermittle die Nektarmenge
4: Bestimme die Wkt. mit der die Futterquellen von beobachtenden Bienen bevorzugt werden
5: Sende die beobachtenden Bienen zu den Futterquellen und ermittle die Nektarmenge
6: Beende die Ausbeutung der erschöpften Quellen
7: Sende die Ausspäher in den Suchraum um neue Futterquellen zufällig zu entdecken
8: Merke die bisher beste Futterquelle
9: **end while**

2.4 Ameisenkolonie-Optimierung

Ameisen erkundschaften zunächst die Umgebung ihres Nestes. Finden sie eine Nahrungsquelle, so bewerten sie diese nach Qualität und Quantität und kehren anschließend zurück zu ihrem Nest. Auf ihrem Rückweg legen sie eine Fährte aus Pheromonen, sodass sie und andere Ameisen zu der Quelle zurückfinden. Die Konzentration an ausgeschiedenen Pheromonen spiegelt dabei die Güte der in der Quelle auffindbaren Nahrung wieder (21). Andere Ameisen wählen mit größerer Wahrscheinlichkeit einen Weg mit höherer Pheromonkonzentration aus. Diese *indirekte* (10) Kommunikation der Ameisen mittels Pheromonen begünstigt das Ausbeuten von *besseren* Quellen und führt dazu, dass die Ameisen den kürzesten Weg zu einer Quelle finden.

2.4.1 Die Metaheuristik

ACO löst Variablenbelegungsprobleme mit Hilfe eines Pheromon-Modells. Die Ameisen starten in ihrem Nest mit einer leeren Lösung und wählen probabilistisch abhängig vom Pheromonwert und der Kosten des Wertes für jede Variable eine Belegung, solange die gefundene Lösung die Nebenbedingungen erfüllt. Wird eine gültige Lösung gefunden, so wird für alle Werte der Lösungen die Wahrscheinlichkeit erhöht, dass diese wieder gewählt werden. Für alle nicht gewählten Werte wird die Auswahlwahrscheinlichkeit verringert, was als Vergessen oder Evaporation bezeichnet wird und die Konvergenz sicherstellt. Dieser Vorgang wird als Pheromon-Update bezeichnet.

2.5 Partikelschwarmoptimierung

> *"The particle swarm algorithm imitates human social behavior."*
> *– Eberhart und Kennedy 1995*

Partikelschwarmoptimierung (PSO) ist eine Metaheuristik, die in ihrer Ursprungsform zur Optimierung von Problemstellungen im kontinuierlichen Suchraum entwickelt wurde. Inzwischen existieren bereits auch Varianten

zur Lösung kombinatorischer Optimierungsproblemen (22). Der Algorithmus wurde 1995 von Eberhart und Kennedy (23) entwickelt und wird neben der Schwarmintelligenz auch häufig zu den Evolutionären Optimierungsalgorithmen gezählt. Im Gegensatz zu den oben vorgestellten Algorithmen basiert PSO nicht auf dem Verhalten von Insekten, sondern ist auf Sozialverhalten von Vogelschwärme, Herden und Menschen zurückzuführen. Eberhart bezeichnete den Algorithmus als eine Imitation des menschlichen Sozialverhaltens.

2.5.1 Idee und Ursprung

Getrieben von der Idee die ästhetische synchrone Bewegung von Vögeln im Schwarm graphisch zu simulieren, entwickelte C. Reynolds (24) das sogenannte *Boids-Modell*. Das Modell beschreibt ein Individuum, als *Boid* bezeichnet, das sich anhand dreier Vorschriften bewegt: *Ausrichtung*, *Kohäsion* und *Trennung*.

Ausrichtung
Jedes Boid im Schwarm versucht sich in die gleiche Richtung wie seine Nachbarn zu bewegen.

Kohäsion
Das Individuum versucht gleichen Abstand zu seinen Nachbarn zu behalten.

Trennung
Jedes Boid versucht eine Richtung zu wählen, die eine Häufung von Boids vermeidet.

Heppner, ein Zoologe, (23) interessierte sich für die Regeln, die es Vögeln im Schwarm ermöglicht, sich synchron zu bewegen, auch bei plötzlichen Richtungswechsel, Streuung und Umgruppierung. Er und Grenander (25) griffen das Modell auf und erweiterten es um sogenannte Rastplätze - eine zusätzliche Eigenschaft, welche die Vögel dazu antreiben soll, auf diesen zu landen. Je näher sich ein Vogel dem Rastplatz nähert, um zu größer steigt der Wunsch dort zu landen.
Auf Grundlage der Arbeiten von Reynolds, Heppner und Grenander entwickelten Eberhart und Kennedy den PSO. Statt wie Heppner und Grenander nach nur einem Rastplatz zu suchen, führen sie den sogenannten Kornfeldvektor ein, in welchem die Vögeln nach dem besten Futterplatz suchen. (23) Jedes Individuum wertet in jeder Generation seine aktuelle Position bezüglich dieses Vektors aus.
Während seiner Suche nach Futter ist jeder Vogel von *inertia*, zu Deusch Trägheit, seiner bisher besten Position, als *simple nostalgia* bezeichnet, sowie der Gemeinschaft oder seinen unmittelbaren Nachbarn beeinflusst, was als *envy* zu Deutsch Neid, bezeichnet wird. *Inertia* ist zu interpretieren, als dass der Vogel träge ist und ungern von guten Futterstellen aufbricht. Die simple Nostalgie deutet an, dass der Vogel sich an seine alten Futterstellen erinnert und dorthin zurückfliegen möchte, wo er am meisten Futter fand.

Im metaphorischen Kontext bedeutet *Neid*, dass der Vogel auch die Futterstellen seiner Nachbarn bzw. der Gemeinschaft kennt und auch von diesen angezogen wird.

2.5.2 *Der PSO-Algorithmus*

Der PSO-Algorithmus modelliert die beschriebene Vogel-Metapher. Der Vogel wird in diesem Kontext als Partikel bezeichnet. Eine Menge von Partikel wird Population genannt. Sei N die Populationsgröße und $i = 1, ..., N$ die Indizes der Partikel.

In jedem Zeitschritt t, auch Generation genannt, ändert ein Partikel X_i seine Position. Diese verändert sich in Abhängigkeit seiner neuen Geschwindigkeit V^{t+1}. Die Geschwindigkeit eines Partikels wird ebenfalls in jeder Generation modifiziert. Analog zur Vogel-Metapher, wird diese durch seine bisher beste Position P_i^t sowie der Position seines bestgelegenen Nachbarn L_i^t (bzw. der bestgelegene der Gemeinschaft G^t), beeinflusst (siehe Abbildung 2.3).

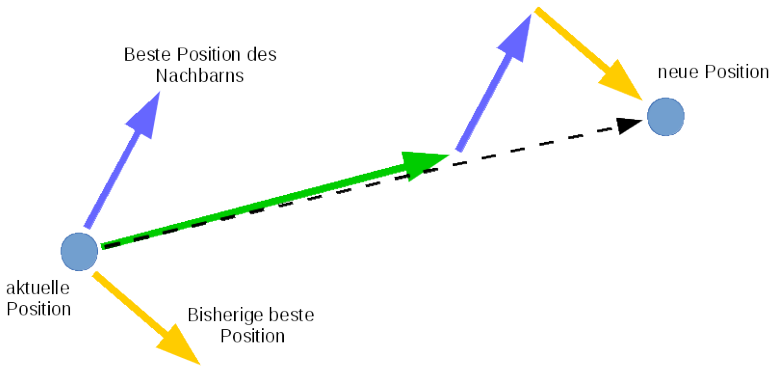

Abbildung 2.3: Visualisierung der Partikelbewegung

Die Berechnungsvorschrift für die neue Geschwindigkeit V_i^{t+1} eines Partikel i in Generation $t+1$, basierend auf seiner aktuellen Position X_i und seiner vorherigen Geschwindigkeit V_i^t (**Trägheit**) sowie seiner bisher besten Position P_i^t (**simple Nostalgie**) und dem Einfluss des bisher besten Partikel der Gemeinschaft[3] G^t (**Neid**) ist in Gleichung (2.1) dargestellt.

[3] der beste positionierte Partikel der Gemeinschaft wird als *gbest* bezeichnet und steht für *global best*

$$V_i^{t+1} = V_i^t + \theta_1 r_1^t (P_i^t - X_i^t) + \theta_2 r_2^t (G^t - X_i^t) \tag{2.1}$$

mit

$$r_1^t, r_2^t \in U(0,1)$$

Gleichung 2.2 zeigt die Aktualisierung der Partikelposition in Abhängigkeit der neuen Geschwindigkeit.

$$X_i^{t+1} = X_i^t + V_i^{t+1} \tag{2.2}$$

Die beiden Komponenten

$$\theta_1 r_1^t (P_i^t - X_i^t) \tag{2.3}$$

und

$$\theta_2 r_2^t (G^t - X_i^t) \tag{2.4}$$

werden als kognitive Komponenten bezeichnet und θ_1, θ_2 als kognitive Lernfaktoren. Die Bedeutung der Lernfaktoren wird im Abschnitt *Paramter* erläutert.

Algorithm 2 PSO - Pseudocode

```
1:  initialisierePopulation()
2:  while not isAbbruchbedingungErfüllt() do
3:     for all particle i = 1...N do
4:        setBestePersönlichePosition()
5:        aktualisiereGlobalBesten()
6:     end for
7:     for all particle i = 1...N do
8:        aktualisiereGeschwindigkeit()
9:        aktualisierePosition()
10:    end for
11: end while
```

Algorithmus 2 zeigt den Pseudocode des grundlegenden Partikelschwarmoptimierers für ein Minimierungsproblem. Zur Verbesserung der Lesbarkeit wurde auf den Superscript t, der die Generation bezeichnet, verzichtet. $g(x)$ stellt eine beliebige, aber feste Gütefunktion dar.

Nach der Initialisierungsphase in Zeile 1, in der jedem Partikel der Population eine zufällige Position zugeordnet und seine Geschwindigkeit mit zufälligen Werten im Wertebereich $[V_{min}, V_{max}]$[4] initialisiert wird, startet der Suchprozess (Zeile 2) , bei der bis zum Eintreten eines Abbruchkriteriums, die Position von jedem Partikel einmal pro Generation nach den Gleichungen (2.1) und (2.2) aktualisiert wird. In jeder Generation wird für jeden Partikel überprüft, ob seine aktuelle Position besser als seine bisher beste ist bzw. ob diese besser als der global beste ist und ggfs. eine Aktualisierung vorgenommen.

[4] Der Wertebereich dient zur Beschränkung der Geschwindigkeit, deren Bedeutung wird im Abschnitt *Parameter* erläutert

2.5.3 Parameter

PSO enthält einige Parameter, mit denen sich das Verhalten des Algorithmus kontrollieren lässt. Der nachfolgende Abschnitt beschäftigt sich mit den Kontrollparametern des Algorithmus.

Partikelschwarmtopologien Unter *Topologie* versteht man beim Partikelschwarmoptimierer die Anordnung der Nachbarn, die einen Partikel beeinflussen. Diese kann sich dabei nach geometrischen Kriterien wie z.B. die am nächsten liegenden Nachbarn zu diesem Partikel oder nach *sozialen* Kriterien (ein fester Subschwarm) richten.

Die ursprüngliche Variante von PSO verwendet eine sogenannte *gbest* Topologie, auch als *all* Topologie bekannt (11). Bei dieser Struktur ist allen Partikel der derzeit global beste Partikel bekannt und alle Partikel richten sich nach diesem. Diese Variante von PSO gilt jedoch als anfällig für lokale Minima. In dem ein Partikel nicht von allen anderen Partikel beeinflusst wird, sondern nur von einem Subschwarm der Größe $\sigma < N$ der Partikelanzahl, lässt sich nach Eberhart diese frühzeitige Konvergenz verhindern. Diese Topologie wird als *lbest*[5] bezeichnet. Bei dieser wird jeder Partikel durch den besten aus seiner Nachbarschaft beeinflusst, statt durch den global besten.

[5] *lbest* steht für local best

Bei den Topologien wird darüber hinaus zwischen dynamischen und statischen Topologien sowie Mischformen aus beiden unterschieden. Als dynamische Topologien werden solche bezeichnet, die sich in jeder Generation ändern . Statische Nachbarschaftsstrukturen dagegen werden zu Beginn des Algorithmus einmalig definiert und bleiben während des gesamten Optimierungsvorgangs erhalten. Auch Mischformen aus beiden existieren. Eine Variante sieht vor, dass die statisch festgelegte Nachbarschaftsstruktur bei Stagnierung des Algorithmus neustrukturiert wird.

STATISCHE UND DYNAMISCHE TOPOLOGIEN

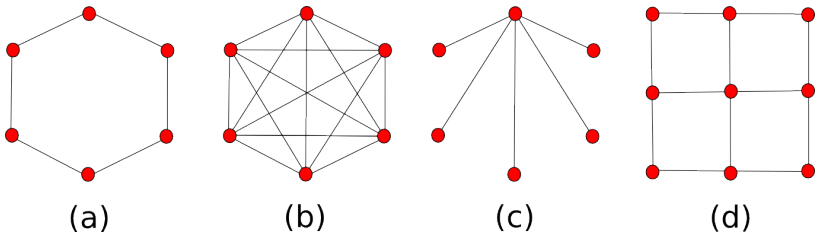

Abbildung 2.4: Gängige Nachbarschaftsbeziehungen

Abbildung 2.4 (a) zeigt eine *ring*-Topologie. Bei dieser ist jedes Partikel zu zwei Nachbarn verbunden. (b) zeigt die sogenannte *all*-Topologie, bei der jedes Partikel zu allen anderen Partikel verbunden ist (Vollständiger Graph). (c) zeigt eine *wheel*-Topologie, bei der ein zentrales Partikel zu allen anderen verbunden ist. Abbildung (d) zeigt eine *square*-Topologie, bei der jedes Partikel zu vier Nachbarn verbunden ist.
Je nach Topologie kann die Performance[6] eines Partikelschwarmoptimierers stark variieren.

[6] Die Performance des PSO bemisst sich an der Konvergenzrate sowie an der Qualität des Fundes bei der Suche

Kognitive und soziale Lernfaktoren Die kognitiven und sozialen Lernfaktoren $\theta_1, \theta_2 \in \mathbb{Z}^+$ beeinflussen wieviel Einfluss der gesamte Schwarm (bei *gbest*) bzw. der Subschwarm eines Partikel (bei *lbest*) sowie seine bisher beste Position auf dessen neue Geschwindigkeit haben. Gilt $\theta_2 > \theta_1$ so tendiert der gesamte Schwarm dazu, sich der Position des bisher besten Partikel *gbest* zu nähern, was zu einem schnelleren Zusammenziehen des Schwarms und somit zu einem detaillierteren Durchsuchen eines bestimmten Bereichs (starke *explotation*) führt.[7] Dieses Suchverhalten führt dabei tendentiell zu einer schlechteren *exploration* des Suchraums. Bei $\theta_1 > \theta_2$ hingegen, wird diese gefördert, der Schwarm erkundet länger den Lösungsraum, ehe der Algorithmus konvergiert. Die Lernfaktoren - *kognitive* und *soziale Lernrate*

[7] selbiges gilt auch bei Nutzung von *lbest*

sind in der Regel Vektoren mit der Dimension des Problemraums. Allerdings gibt es auch Varianten die stattdessen einen Skalar verwenden, wie es z.B. beim *linear PSO (L-PSO)* der Fall ist (26).

L-PSO

Beschränkung der Geschwindigkeit Die Geschwindigkeit gibt an, mit welcher Granularität der Suchraum zwischen der aktuellen Position eines Partikel und dem Ziel, der bisher besten Position, durchsucht wird. Wird die Geschwindigkeit nicht begrenzt, so kann es passieren, dass die Partikel wild umher irren und der Algorithmus nicht konvergiert. Aus diesem Grund wird eine obere (bzw. untere) Begrenzung v_{max} (bzw. v_{min}) verwendet.

v_{max} ist ein Skalar, das eine obere Grenze für die aufsummierten Einzelwerte des Geschwindigkeitsvektors darstellt. Alternativ lässt sich jedoch auch ein Vektor V_{max} verwenden, der an Stelle i die obere Schranke des i-ten Geschwindigkeitseintrags enthält. (v_{min} analog)
Der Wert von v_{max} (bzw. v_{min}) ist dabei entscheidend. Ein zu hoher Wert kann dazu führen, dass gute Lösungen übersprungen werden, während ein zu niedriger Wert zur frühzeitigen Konvergenz führen kann, da ein lokales Minimum nicht übersprungen wird. Als Werte für die Geschwindigkeitsgrenzen empfiehlt Shi 10%-20% des Suchraumbereichs(27).

Inertia Weight Einen weiteren Einfluss auf die Performance des PSO kann die Nutzung eines sogenannten *inertia weights* ω, zu Deutsch Trägheitsgewicht, haben. Dieses hat einen Einfluss auf den Trade-Off zwischen *exploration* [8] und *explotation*[9]. Das Trägheitsgewicht bestimmt, wieviel Einfluss die Trägheit, also die vorherige Geschwindigkeit auf die aktuelle Geschwindigkeit hat. Gleichung (2.5) zeigt die Berechnung der neuen Geschwindigkeit unter Verwendung eines Trägheitsgewichts.

[8] Erkunden des gesamten Suchraums, verfolgt das Ziel so breitflächig wie möglich den Suchraum zu erkunden
[9] Feingranulares Durchsuchen des Lösungsraums

$$V_i^{t+1} = \omega V_i^t + \theta_1 r_1^t (P_i^t - X_i^t) + \theta_2 r_2^t (G^t - X_i^t) \tag{2.5}$$

Nach Shi und Eberhart (28) begünstigt ein hohes Trägheitsgewicht die globale und ein niedriges die lokale Suche. Mittlerweile wurden unterschiedliche Varianten zur Berechnung von Trägheitsgewichten vorgeschlagen. Einen Überblick über verschiedene solcher Strategien liefert Bansal in seiner Publikation *Inertia Weight Strategies in Particle Swarm Optimization* (29).

Constriction Coefficient Während das Trägheitsgewicht ausschließlich Einfluss auf die Trägheit ausübt, ist *constriction coefficient* ein Faktor, der auch die *soziale* und *kognitive* Komponente skaliert. Ein zu kleiner Wert des Faktors führt dazu, dass der Suchraum evtl. nicht vollständig durchsucht wird, wohingegen ein zu hoher Wert, die Konvergenzrate erhöht, was jedoch zu einer besseren explotation führt. Die Berechnung des *constriction*-Faktors χ ist in Gleichung (2.6) dargestellt.

$$\chi = \frac{2}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{2.6}$$

mit

$$\phi = \theta_1 + \theta_2, \phi > 4$$

Beschränkung des Suchraums Ist der Suchraum bei einem Optimierungsproblem beschränkt, lassen sich die Werte, die eine Variable annehmen kann, durch die Vektoren $X_{min} \in \mathbb{R}^d$ und $X_{max} \in \mathbb{R}^d$ kontrollieren, wobei X_{min_i} den niedrigsten und X_{max_i} den höchsten Wert darstellen, den ein Partikel im i-ten Eintrag annehmen kann.

Nach Aktualisierung eines Partikel lässt sich eine Unter- bzw. Überschreitung des zulässigen Wertebereichs durch unterschiedliche Methoden korrigieren. Es folgen drei Beispielmethoden:

1. Stick
   Bei dieser Methode wird bei Überschreiten der Dimensionsgrenze die entsprechende Variable genau auf den höchstmöglichen Wert gesetzt. (Unterschreitung analog)

2. Bounce
   Bei der *bounce*-Methode springt der Partikel bei der jeweiligen Variable um die Differenz zwischen der neu berechneten Position und der Dimensionsgrenze in den gültigen Wertebereich zurück.

3. Wrap
   Bei *wrap* wird der Suchraum als ein Ring dargestellt. Überschreitet ein Partikel die Dimensionsgrenze an der oberen Schranke, so entspricht der Wert der jeweiligen Variable der unteren Dimensionsgrenze, erweitert um die Differenz zwischen der überschrittenen Dimensionsgrenze und der neu berechneten Position.[10] (Unterschreitung analog)

[10] bei Überschreitung implementierbar durch eine Modulo-Operation

3
Lösungsansatz

3.1 Formulierung der Zielfunktion

Die Grundidee zur Lösung des oben beschriebenen Optimierungsproblems besteht darin, das Ballmodell mit mehreren unterschiedlichen Parametersätzen durchlaufen zu lassen und den Parametersatz mit dem geringsten Fehler unter diesen auszuwählen. Im Ballmodell wird also für die Rauschmatrix des erweiterten Kalman Filters ein Parametersatz $x \in \mathbb{R}^n$ gesucht, so dass der Erkennungsfehler $f(x)$ minimiert wird, d.h.

$$x = \underset{x_i}{\operatorname{argmin}} f(x_i)$$

3.2 Wahl des Algorithmus

"There ain't no such thing as a free lunch"

– Wolpert und Macready 1997

Gemäß des oben zitierten *No free lunch theorem* gibt es nicht *das* universelle, beste Verfahren, das alle Probleme am besten löst, sondern unterschiedliche Verfahren, die sich unterschiedlich gut für ein Problem eignen. Es gilt das richtige Verfahren für sein Problem auszuwählen.

Aufgrund der im Abschnitt *Schwarmintelligenz* aufgezählten Vorteile der SI-Verfahren wurden zur Lösung des beschriebenen Optimierungsproblems unterschiedliche Verfahren aus der Kategorie der Schwarmintelligenz herangezogen. Aus den drei betrachteten Algorithmen wurden ACO und PSO als Lösungsverfahren in Erwägung gezogen, da diese eine bessere Performance haben. ACO ist zwar als kombinatorischer Optimierungsalgorithmus für das Lösen im diskreten Lösungsraum entworfen worden, jedoch wurde vom Entwickler des Ameisenkolonie-Optimierungsalgorithmus, M. Dorigo eine Variante für die kontinuierliche Domäne entwickelt(30). Der $ACO_{\mathbb{R}}$, wie dieser abgekürzt wird, erzielte im von Liao et al. (31) im Jahr 2012 durchgeführten Benchmark bessere Ergebnisse für einfache und schlecht konditionierte Funktionen[1] sowie für multimodale[2] Funktionen im Vergleich zu den Ergebnissen von PSO, GA (Genetischer Algorithmus) und ABC aus früheren Benchmark Workshops. Somit kämen sowohl PSO als auch ACO in

[1] In der numerischen Mathematik ist eine Konditionszahl eine Kennzahl, welche Auskunft darüber gibt, wie stark sich die Ausgabe einer Funktion bei Veränderung der Eingabedaten ändert. Schlecht konditioniert drückt aus, dass eine geringe Änderung der Eingabedaten einen verhältnismäßig starken Einfluss auf die Ausgabe hat.

[2] Funktionen mit mehreren Optima

Frage. Letztendlich verwende ich den PSO da dieser bereits auf das Problem der Schätzung von Extended Kalman Filter-Kovarianzmatrizen erfolgreich angewandt wurde (3). Ein weiterer, wenngleich nebensächlicher Vorteil ist die verhältnismäßig geringe Parameterzahl[3] , die es im Vergleich zu anderen Schwarmintelligenz-Verfahren, bei diesem Algorithmus zu justieren gilt. Nichtzuletzt hat sich dieses Verfahren auch im Zusammenhang mit Parameteroptimierung im Umfeld von humanoiden Fußball spielenden Robotern mehrmals bewährt und bietet daher für das hier vorgestellte Problem einen vielversprechenden Ansatz (9), (8).

[3] Die Parametrisierung von Algorithmen ist immer problemspezifisch. Die Bestimmung einer guten Konfiguration wird umso aufwendiger, je mehr Parameter zu setzen sind

3.2.1 Bestimmung geeigneter PSO-Variante

Mittlerweile gibt es eine Vielzahl an PSO-Varianten. Zur Bestimmung einer geeigneten unter diesen für das zu lösende Problem, wurde eine Auswahl unterschiedlicher Varianten getroffen und diese getestet.

3.2.2 Selektion der Varianten

Bei der Wahl der zu vergleichenden Varianten habe ich mich an der Publikation von Bansal aus dem Jahr 2011 orientiert (29), in welcher er 15 unterschiedliche Trägheitsgewichtsfunktionen mit den Benchmark-Funktionen *Sphere*, *Griewank*, *Rosenbrock*, *Rastrigin* sowie *Ackley* testet. Eine Zusammenfassung seiner Ergebnisse ist auf der Tabelle in Abbildung 3.1 zu sehen.

Criterion	Best Inertia Weight Strategy	Worst Inertia Weight Strategy
Average Error	Chaotic Inertia Weight	Chaotic Random Inertia Weight
Average Number of Iterations	Random Inertia Weight	Constant Inertia Weight
Minimum Error	Constant Inertia Weight Linear decreasing Inertia Weight	Chaotic Random Inertia Weight Global-Local Best Inertia Weight

– Bansal, J. (29)

Abbildung 3.1: Zusammenfassung - Benchmark Trägheitsgewichtsfunktionen

Aus der Tabelle gehen die am besten bzw. am schlechtesten geeigneten Trägheitsstrategien bzgl. der statistischen Größen *Durchschnittsfehler*, *Durchschnittsiterationszahl* sowie *kleinster Fehler* hervor.

Wie aus dieser erkennbar ist, liefert die sogenannte *linear decreasing inertia weight*-Strategie (siehe Gleichung (3.2)) neben dem *constant inertia weight* PSO den kleinsten Fehler. *constant inertia weight* liefert jedoch auch die höchste durchschnittliche Iterationszahl. Aus diesem Grund scheint erstere Strategie sinnvoller. Da es sich bei *constant inertia weight* lediglich um eine Konstante vor der *velocity* aus der vorherigen Periode handelt, lässt sich diese Variante effizient berechnen, weshalb ich diese ebenfalls selektiert ha-

LINEAR DECREASING INERTIA WEIGHT

be.

$$\omega_t = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{gen_{max}} \cdot t \tag{3.2}$$

wobei ω_t das Gewicht in Generation t bezeichnet, ω_{min} und ω_{max} die Minimal- und Maximalgewichte darstellen und gen_{max} die maximale Generationszahl darstellt.

Bei der dritten getesteten Variante handelt es sich um die *random inertia weight* Strategie (siehe Gleichung (3.3)), welche sich nach Bansal (29) am besten für Effizientes Durchsuchen des Lösungsraums eignet und wie aus der Tabelle zu entnehmen ist, die kleinste durchschnittliche Iterationszahl enthält.

RANDOM INERTIA WEIGHT

$$\omega = 0.5 + \frac{Rand()}{2} \tag{3.3}$$

wobei Rand() eine reelle Zahl im offenen Intervall (0,1) liefert.

3.2.3 Erste Tests

Um das Verhalten der Varianten *RIW-*, *LDIW-* und *CIW-PSO* bzgl. der zu optimierenden Zielfunktion zu verstehen, wurden diese mit der in Tabelle 3.1 aufgeführten Konfiguration in drei Durchläufen getestet. Die verwendete Topologie entspricht der *lbest*.

Nachbarschaftsgröße	4
Populationsgröße	20
Generationen	40
θ_1	3
θ_2	5
Wertebereich *friction*	$[-0.5, -0.01]$
Wertebereich *process_update.invalid_error_x_threshold*	$[100, 1000]$
Wertebereich *process_update.invalid_error_y_threshold*	$[100, 1000]$
Wertebereich *process_update.noise_pos*	$[1e-08, 1000]$
Wertebereich *process_update.noise_vel*	$[1e-08, 1000]$
Wertebereich *sensor_update.noisepitch*	$[1e-08, 1000]$
Wertebereich *sensor_update.noiseyaw*	$[1e-08, 1000]$
Geschwindigkeit *friction*	$[-0.05, 0.05]$
Geschwindigkeit *process_update.invalid_error_x_threshold*	$[-200, 200]$
Geschwindigkeit *process_update.invalid_error_y_threshold*	$[-200, 200]$
Geschwindigkeit *process_update.noise_pos*	$[-200, 200]$
Geschwindigkeit *process_update.noise_vel*	$[-200, 200]$
Geschwindigkeit *sensor_update.noisepitch*	$[-200, 200]$
Geschwindigkeit *sensor_update.noiseyaw*	$[-200, 200]$

Tabelle 3.1: Konfiguration von PSO

Nach dem Vorbild von Bansal wurde als Konstante für *CIW-PSO* ω=0.7 und für *LDIW-PSO* als Minimal- und Maximalgewichte ω_{min}=0.4 und ω_{max}=0.9 gewählt. Die Tests zeigten, dass alle drei Varianten bei der Zielfunktion

schnell stagnieren. Der Versuch das Stagnationsverhalten durch die Verwendung anderer Lernfaktoren (θ_1=2,05 und θ_2=2,05)[4] bzw. (θ_1=5 und θ_2=3) positiv zu beeinflussen, führte im Falle der ersteren nur zu mäßigem Erfolg, während mit letzteren die Stagnation sogar früher eintrat. Die Ergebnisse der unterschiedlichen Durchläufe können Tabelle 3.2 entnommen werden.

SCHNELLE STAGNATION

[4] wie in (32) empfohlen

LF	CIW	CIW-K	LDIW	LDIW-K	RIW	RIW-K
θ_1=2,05	258,49	33	260,20	17	258,52	20
θ_2=2,05	258,49	27	258,54	33	258,47	36
	258,70	29	258,54	30	258,52	26
θ_1=5	258,85	8	258,90	11	258,60	2
θ_2=3	259,09	3	258,78	21	258,77	29
	286,64	8	258,87	12	260,26	20
θ_1=3	258,78	16	258,79	16	258,88	34
θ_2=5	258,74	14	258,81	9	259,00	9
	258,71	15	258,73	26	258,81	15

Tabelle 3.2: Ergebnisse der Testdurchläufe

Des Weiteren wurde versucht, durch Neuzuweisung der Nachbarschaften den Schwarm "neuzubeleben", sobald nach mehreren Generationen keine Verbesserung des besten Partikel eintrat, was allerdings auch nur bedingt Verbesserung brachte. Die besten Resultate konnten mit einer anderen PSO-Variante erzielt werden, die den Zustand von Stagnation oder frühzeitiger Konvergenz identifiziert und behandelt, der sogenannten *RSVN-p-PSO* (32). Das Kürzel *RSVN* steht für ***R**andom **S**ampling in **V**ariable **N**eighboorhoods* und bezeichnet eine Sampling-Strategie, welche eine gute Streuung der Partikel ermöglicht, ohne sich dabei gänzlich von bereits gefundenen, viel versprechenden Regionen im Suchraum zu entfernen. Dies geschieht durch das Sampling von Nachbarschaften um den globalen besten herum. Die genaue Vorschrift nach (32) kann den Gleichungen (3.4) - (3.8) entnommen werden. M bezeichnet dabei die Anzahl an Nachbarschaften, d stellt den Index der Dimension dar, $\zeta_j = j/M$ ist der sogenannte Nachbarschaftsfaktor und stellt das Verhältnis von j-ter Nachbarschaft und Größe des Suchraums dar, Ψ_j ist die j-te Menge an gleichverteilt gesampelten Positionen aus dem Intervall $[\lambda^-_{jd}, \lambda^+_{jd}]$.

NEUZUWEISUNG DER NACHBARN

BEHANDLUNG VON STAGNATION DURCH RESAMPLING

$$\lambda^-_{jd} = \begin{cases} \tau_{jd}, \tau_{jd} \geq \sigma_{min} \\ \sigma_{min}, \tau_{jd} < \sigma_{min} \end{cases} \tag{3.4}$$

mit

$$\tau_{jd} = G_d - \zeta_j |\sigma_{max} - \sigma_{min}| \tag{3.5}$$

$$\lambda^+_{jd} = \begin{cases} \gamma_{jd}, \gamma_{jd} \leq \sigma_{max} \\ \sigma_{max}, \gamma_{jd} > \sigma_{max} \end{cases} \tag{3.6}$$

mit

$$\gamma_{jd} = G_d + \zeta_j |\sigma_{max} - \sigma_{min}| \tag{3.7}$$

$$X_t \in \Psi_j | X_{td} \sim U(\lambda_{jd}^-, \lambda_{jd}^+), t = 1, ..., |\Psi_j|; j = 1, ..., M \tag{3.8}$$

Die von den Autoren präsentierte PSO-Variante entspricht einem *gbest* (mit *all*-Topologie), der zusätzlich zu der *linear decreasing inertia weight*-Strategie einen *Constriction*-Faktor verwendet. Nach dem ein gesamter Zyklus durchlaufen wurde, wird überprüft, ob die zulässige Höchstzahl an Generationen ohne Fortschritt überschritten ist. Ist dies der Fall, so werden alle Partikel neu gesampelt und der globale beste ggfs. aktualisiert. Nachfolgend wird dieser Schwellenwert als *MaxStagnation* bezeichnet.

3.3 Optimierungsprozess

Abstraktes Modell Nach Initialisierung eines Parametersatzes mit Hilfe einer beliebigen, aber festen Verteilungsfunktion wird mit einer Suchheuristik wiederholend, ausgehend von der aktuellen Position, die nächste Position bestimmt. Auf diese Art und Weise wird der Parameterraum erkundet. Dabei wird jede besuchte Position im Raum mit einer Gütefunktion ausgewertet. Nach mehreren Iterationsschritten mit unterschiedlichen Parametersätzen werden die getesteten Parametersätze verglichen und der beste darunter als Ergebnis geliefert. Abbildung 3.2 zeigt eine vereinfachte Darstellung des beschriebenen Modells.

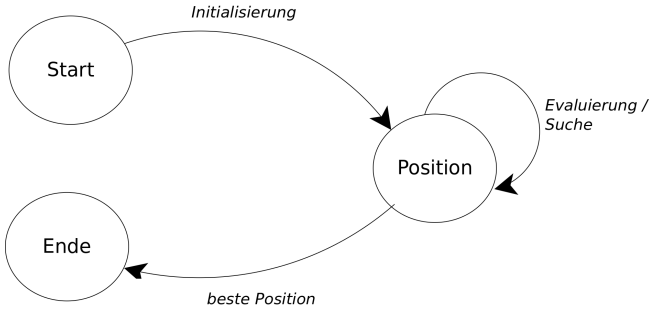

Abbildung 3.2: Einfaches Optimierungsmodell

Optimierung mit dem Ballmodell Das obige abstrakte Modell auf das Problem des Ballmodells übertragen, sieht wie folgt aus: Nach der Initialisierung des PSO-Schwarms mit zufälligen oder festgelegten Werten für jedes Partikel, wird der erweiterte Kalman Filter mit jedem Partikel (Parametersatz) einmal ausgeführt. Für jedes Partikel liefert der Kalman Filter eine Folge von Schätzungen[5] der relativen Position des Balles bzgl. des Roboters auf dem Spielfeld. Mit Hilfe einer Gütefunktion und gelieferten Groundtruthdaten wird die mittlere Abweichung über alle Frames der Schätzungen von der Groundtruth und somit die Güte eines jeden Partikel bestimmt. Die berechneten Gütewerte werden an den Partikelschwarmoptimierer geliefert, auf Grundlage derer er neue Positionen für die Partikel berechnet. Dieser Vorgang wird wiederholt ausgeführt und terminiert, sobald die vordefinierte

[5] Alle Informationen, die zur Berechnung einer Schätzung erforderlich sind, werden in so genannte Datenframes (kurz *frames*) gekapselt

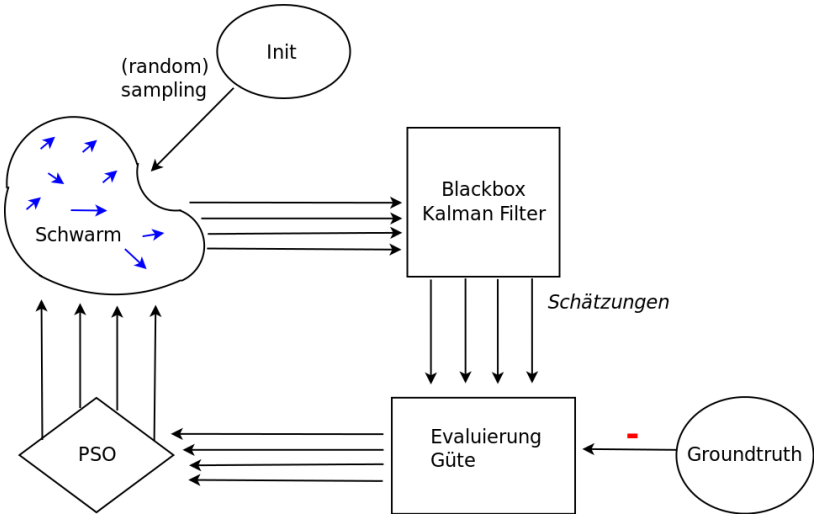

Abbildung 3.3: Optimierungsprozess

Höchstzahl an Iterationen erreicht oder Stagnation bzw. Konvergenz eingetreten ist. Abbildung 3.3 illustriert den Arbeitsablauf.

GÜTEFUNKTION

Die Gütefunktion muss die Groundtruthdaten berücksichtigen, um eine sinnvolle Aussage über die Qualität eines Parametersatzes treffen zu können. Zu diesem Zweck verwende ich ein Globales Tracking System wie von Burchardt (9) beschrieben, welches Auskunft über die tatsächliche absolute Position des Balles p_i für jedes Frame f_i liefert. Die gelieferten Groundtruthdaten lassen sich dann mit den geschätzten Positionen[6] für das jeweilige Frame vergleichen und daraus eine Gütefunktion für die Datenmenge bestimmen.

[6] Die Umwandlung der absoluten Koordinaten in relative Koordinaten bzgl. des Roboters ist dazu erforderlich.

VERGLEICHBARKEIT DER PARAMETERSÄTZE

Darüber hinaus muss sichergestellt sein, dass die Umgebung, in der unterschiedliche Parametersätze getestet werden, vergleichbar ist. Konkret bedeutet dies, dass die Eingabedaten, die das Ballmodell zusätzlich zu den Rauschmatrizenparametern verwendet, identisch sein müssen. Diese Anforderung macht es erforderlich, dass die Sensordaten des Roboters und die daraus extrahierten Informationen, die das Ballmodell zur Berechnung benötigt, aufgezeichnet werden und mit jeden Parametersatz erneut abgespielt werden kann. S. Otte implementierte in seiner Masterarbeit den sogenannten LogPlayer, der das Aufzeichnen solcher in Datenframes gekapselten Prozessdaten sowie das Abspielen dieser ermöglicht. Ein solcher Satz von Datenframes bezeichnete er als Logdatei.

4
Umsetzung

4.1 Vorbereitung

Im Folgenden werden die für die Experimente erforderlichen Vorbereitungen sowie die Gütefunktion beschrieben.

4.1.1 Logdateien

Für die Optimierung wurden mit Hilfe des erwähnten LogPlayers zwei Logdateien mit je 8.500 Frames mit einem Roboter aufgezeichnet. Dazu wurde der bisher beste manuell justierte Parametersatz verwendet (siehe Tabelle 4.1).

Parametersatz
$[-0,15; 1000; 1000; 5; 10; 0,001; 0,01]$

Tabelle 4.1: Bester manuell bestimmter Parametersatz

Die eine Logdatei, nachfolgend als Trainingsdatum bezeichnet, dient dazu, verbesserte Parameter zu trainieren, während die andere, als Validierungsdatum bezeichnet, zur Validierung des Ergebnisses dient. Das vom Roboter beobachtete Szenario ist wie zwei menschliche Spieler sich gegenseitig den Ball auf einer Spielfeldhälfte zuspielen. Dabei versuchen die Spieler möglichst die gesamte Spielfeldhälfte auszunutzen.

Um das Verhalten des Roboters nicht unbewusst zu beeinflussen, versuchen die Spieler dessen Verhalten zu ignorieren. Darüber hinaus wird versucht, durch längere Reaktionszeit der Spieler und langsamere Bewegung sowie durch kürzere Schritte und einer reduzierten Schusskraft, das Spielverhalten von Roboter zu simulieren.

Fehlerhafte Schätzungen Bei der Inspektion der Logdateien stellte sich heraus, dass einige sehr hohe Abweichungen bei den Schätzungen vorlagen. (siehe Abbildung 4.1).
Histogramm 4.2 zeigt, dass eine Vielzahl der Abweichungen beim Trainingsdatum bei mehr als 1500 mm liegt. Da hohe Schätzfehler in verhältnismäßig großer Zahl auftreten, ist damit zu rechnen, dass diese während der Trainingsphase, bei der Berechnung der Güte der Parametersätze, stark ins Gewicht fallen, was dazu führen dürfte, dass auf diese Ausreißer gelernt

OVERFITTING

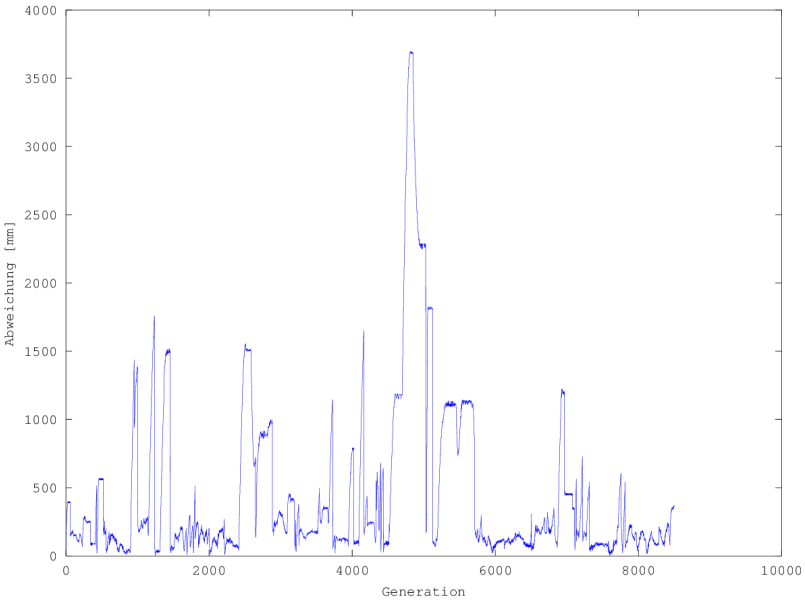

Abbildung 4.1: Abweichung pro Datenframe - vor der Filterung

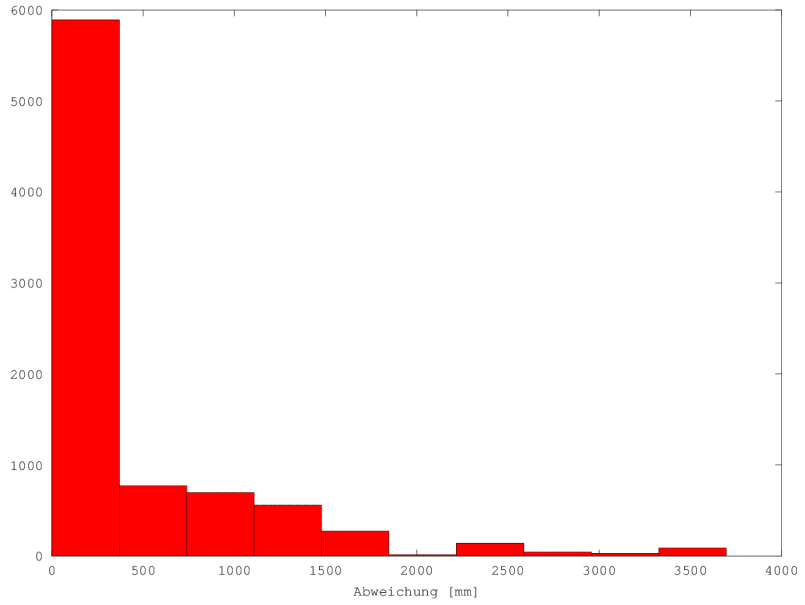

Abbildung 4.2: Fehlerverteilung mit dem manuellen Parametersatz bei Logdatei A

wird.

Eine nähere Betrachtung der gespeicherten Daten in den Logdateien zeigte, dass die hohen Fehlschätzungen auf auftretende Richtungswechsel bzw. Geschwindigkeitswechsel zurückzuführen sind, die der Roboter nicht wahrgenommen hat. Dem Roboter ist es damit nicht möglich eine korrekte Schätzung zu liefern. Die Modellierung eines solchen Szenarios ist nicht möglich, weshalb Sequenzen mit einem solchen Ereignis in beiden Logdateien entfernt wurden. Abbildung 4.3 zeigt die Logdatei nach der Filterung solcher Sequenzen.

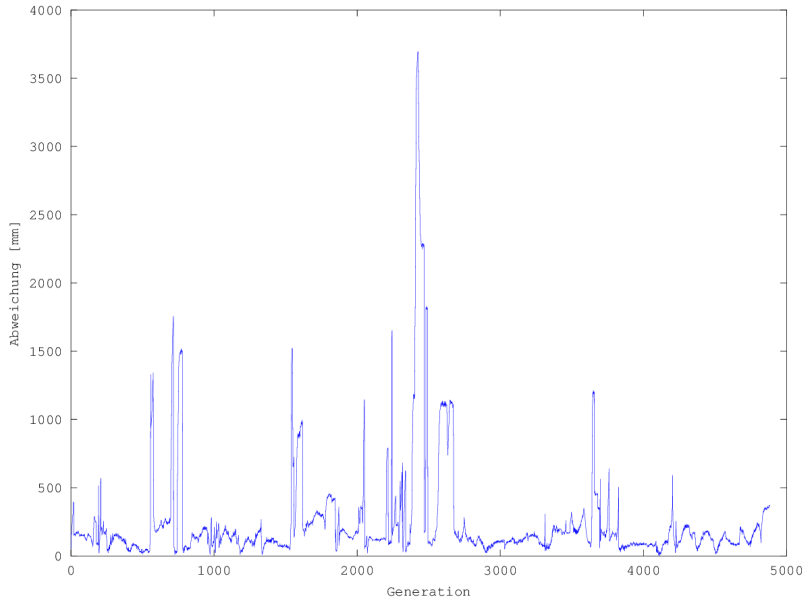

Abbildung 4.3: Abweichung pro Datenframe nach der Filterung

4.1.2 *Groundtruthdaten*

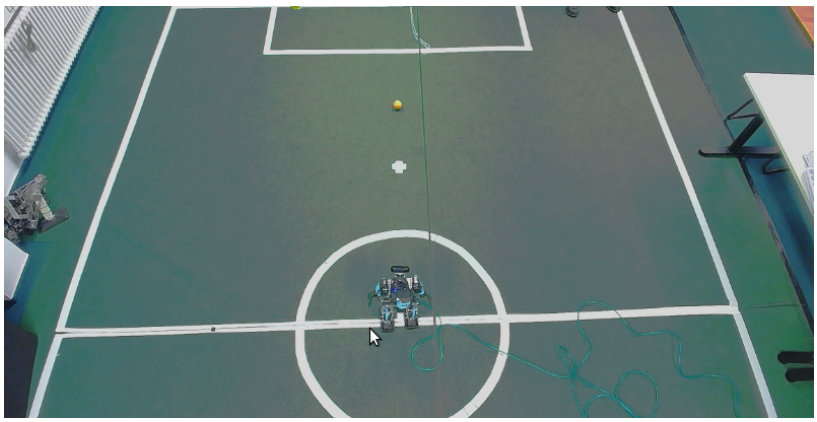

Abbildung 4.4: Sicht aus der Perspektive der Groundtruthkamera

Global Tracking System Nach dem Vorbild von Burchardt wird ein Groundtruthsystem verwendet, um die Güte von Parametersätzen zu bestimmen (9). Die über dem Feld montierte Kamera wird dazu genutzt, den Ball zu tracken. Der Roboter wird auf den Mittelpunkt des Feldes plaziert, wodurch dessen Position implizit gegeben ist. Um die tatsächliche Position des Balles möglichst korrekt tracken zu können, muss die Kamera auf das tatsächliche Spielfeld und die Farben kalibriert werden.

GLOBAL TRACKING SYSTEM

Feld- und Farbkalibrierung der Groundtruthkamera Zur Kalibrierung der Groundtruthkamera wurde die Software *Shared Vision System (SSL-Vision)* verwendet. Diese bietet eine Schnittstelle um Farb- sowie Feldkalibrierung (siehe Abbildung 4.5) durchzuführen. Obwohl eine gute Kalibrierung mit diesem Tool erreicht werden kann, ist aufgrund der niedrigen Auflösung

HTTP://CODE.GOOGLE.COM/P/SSL-VISION/

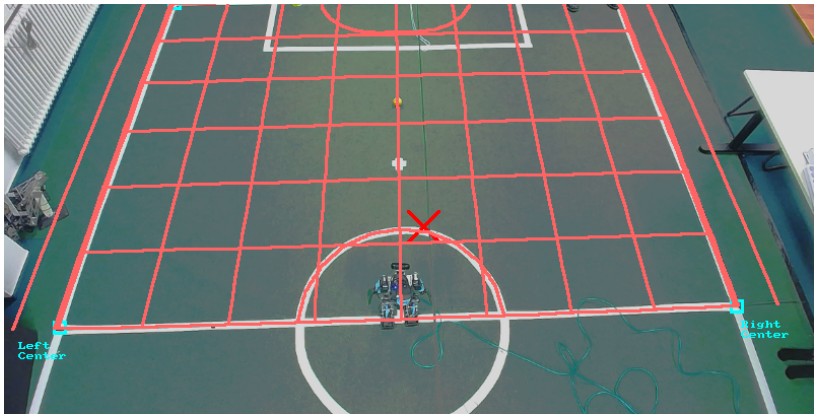

Abbildung 4.5: Feldkalibrierung der Groundtruthkamera

der Groundtruthkamera und sein Blickwinkel auf das Feld eine exakte Positionsbestimmung des Balles kaum möglich. In der Regel liegt eine Abweichung von etwa ± 100 mm vor.

4.1.3 *Gütefunktion*

Die Abweichung zwischen der berechneten Schätzung der Ballposition E_i basierend auf ein Datenframe f_i und der tatsächlichen Position G_i wird mit dem Euklidischen Abstandsmaß bestimmt. Zur Bewertung eines Parametersatzes werden die Abweichung für alle Frames einer Logdatei berechnet und gemittelt. Dabei werden invalide Schätzungen oder Groundtruthdaten nach dem Vorbild von Burchardt nicht berücksichtigt.

4.2 *Implementierung*

Im Folgenden wird die Architektur der verwendeten Framework der FUmanoids vorgestellt und beschrieben wie dieses zur Lösung des Optimierungsproblem erweitert wurde.

4.2.1 *Framework*

Das Hauptframework bei den FUmanoids ist das gleichnamige Roboterprogramm *FUmanoid*, ein *Blackboard* System, welches Berechnungskomponenten, als *Module* bezeichnet, von den Daten, *Repräsentationen*, trennt. Die Hauptkomponente ist das sogenannte Blackboard, ein Objekt, über das Module auf alle Repräsentationen zugreifen können.

FUMANOID

Jedes Module kann seine Abhängigkeiten angeben, aus der sich die Reihenfolge bestimmt, in welcher diese Module abgearbeitet werden.Der Aufruf der jeweiligen Module erfolgt durch die sogenannten *Module Manager*, *Cognition Module Manager* und *Motion Module Manager*. Für das Aufrufen der Module aus der Kategorie Computer Vision und den Modellen ist der sogenannte *Cognition Module Manager* verantwortlich.

4.2.2 *Optimierungsarchitektur*

Architektur Die Optimierungsarchitektur besteht aus zwei Programmen - ein Python Server, der den Optimierungsprozess initialisiert und den Op-

timierungsalgorithmus ausführt und das FUmanoid Framework mit einem zusätzlichen Optimierungsmodul, welches die spezifierte Logdatei auf den relevanten Programmteilen *EkfBallmodeling*[1], *BallPositionProvider*[2] und *GroundTruthProvider*[3] unter Verwendung des LogPlayers abspielt und zwei Vektoren von Ergebnisdaten (Schätzung und Groundtruth) mittels Unix Domain Socket an den Optimierungsserver sendet.

[1] führt die Berechnung des Ballmodells aus

[2] bündelt geschätzte und tatsächliche Position und stellt diese in der gleichen Einheit zur Verfügung

[3] empfängt die Daten vom Groundtruthsystem und stellt diese für das restliche System bereit

PSO-Server Der PSO-Server führt den PSO-Algorithmus aus. Weitere Zuständigkeiten des Servers sind das Starten von FUmanoid-Instanzen mit diesen Parametersätzen sowie das Empfangen und Verarbeiten der Ergebnisvektoren mit Hilfe eines vordefinierten Fehlermaßes. Des Weiteren verwaltet der Server auch die gestarteten Prozesse und sorgt dafür, dass fehlgeschlagene FUmanoid-Instanzen beendet (*gekillt*) und eine neue Instanz mit den selben Parametersätzen neu gestartet werden, wenn diese bis zu einem spezifizierten Timeout das Ergebnis noch nicht geliefert haben.

FUmanoid Das FUmanoid Framework wurde um ein zusätzliches Modul erweitert, welches das Ballmodell mit dem jeweiligen Parametersatz ausführt und die für das Ballmodell wichtigen Daten bereitstellt. Das Modul verwaltet zwei sortierte Vektoren, eins für die Groundtruthdaten und ein weiteres für die Schätzungen. Die Schätzung und die tatsächliche Position des Balles werden, nachdem ein Frame abgearbeitet wurde, den Vektoren hinzugefügt. Sobald alle Frames der Logdatei vollständig abgespielt wurde, überführt das Modul die zwei Vektoren in das Json Format und verschickt das Ergebnisdatum über Unix Domain Socket an den PSO-Server. Anschließend beendet sich das FUmanoid-Programm.

4.3 Parallelisierung

> *"A little step for one cluster, a big step for one core."*
>
> *– O. Morillo*

Die Optimierung fand zunächst auf einem einzigen CPU-Kern mit der oben beschriebenen Variante des Frameworks statt. Da die Berechnung bei der globalen Optimierung jedoch sehr zeitintensiv waren, wurde das Optimierungs-Framework dahingehend angepasst, dass die Kommunikation zwischen FUmanoid und dem PSO-Server sowie die Berechnung des Ballmodells basierend auf ein Partikel parallel ausführbar ist. Die der Implementierung zugrunde liegende Architektur entspricht dem Master-Worker Pattern und ist mit Hilfe der *Messaging* Bibliothek *ZeroMQ* umgesetzt. Die Serialisierung erfolgte mit Hilfe Google's Serialisierunsframework *Google Protocol Buffers* , welches eine Möglichkeit bietet, strukturierte Daten zu serialisieren.

WWW.ZEROMQ.ORG/

WWW.CODE.GOOGLE.COM/P/PROTOBUF/

Der Master ist dabei zuständig für das Berechnen der neuen Partikelpositionen auf Grundlage der Güte der vorherigen Positionen sowie für das Beliefern der Worker mit den Positionen der Partikel. Jeder Worker startet anschließend mit der gelieferten Partikelposition (Parametersatz) eine FUmanoid-Instanz. Nachdem Estimation- und Groundtruthvektoren bei dem Worker eingegangen sind, liefert der Worker diese gemeinsam mit dem Partikel-*Identifier* an den Master zurück.

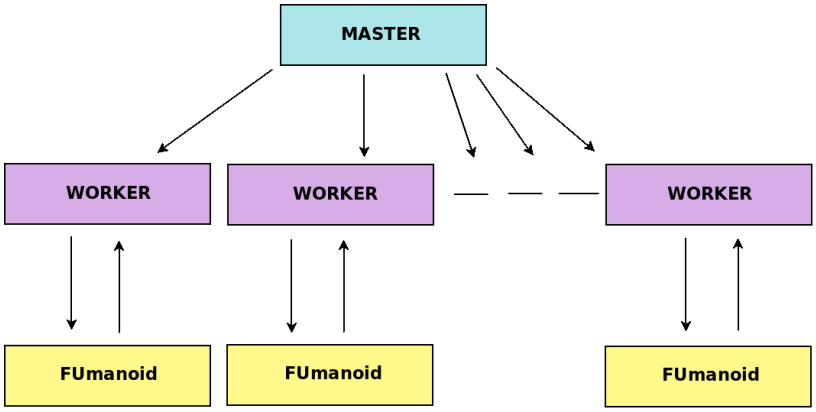

Abbildung 4.6: Master-Worker Pattern

Die neue Architektur wurde auf einem Cluster mit unterschiedlicher Knotenzahl (5-11 Knoten) getestet bei einer Populationsgröße von 800. Der Geschwindigkeitszuwachs kann Tabelle 4.2 entnommen werden.

master	worker	CPU-Kerne	Partikel/Min	SpeedUp
1	0	01	17	1.0
1	4	31	136	8.0
1	6	43	236	13,9
1	8	55	468	27,5
1	10	67	454	26,7

Tabelle 4.2: Überblick des Geschwindigkeitszuwachs in Abhängigkeit der verwendenten Anzahl an CPU-Kerne

Tabelle 4.2 zeigt, dass die Parallelisierung der Ballmodellberechnung einen deutlichen Geschwindigkeitsvorteil liefert. Das Framework skaliert linear in der Anzahl der Kerne. Wie aus der Tabelle deutlich wird, skaliert die Applikation bei 67 Kerne nicht mehr. Dieses ist auf den Master zurückzuführen, der den *bottle neck* darstellt. Dem Master ist es nicht möglich, die von den Worker gelieferten Daten schnell genug zu verarbeiten. Um dieses Problem zu beseitigen ließen sich die vom Master durchgeführten Berechnungen ebenfalls verteilt ausführen.

5
Experimente und Evaluierung

Im folgenden Kapitel wird das unter *Problemstellung* formulierte Ziel der Optimierung aufgegriffen und präzisiert und die Experimente und Ergebnisse beschrieben.

5.1 Ziel und Erwartung

Das Ziel der Optimierung ist die Bestimmung von guten Parametersätzen. Da jedoch keine Eigenschaften der Funktion bekannt sind, lässt sich nicht bestimmen, ob ein gegebener Parametersatz das globale Optimum für die Zielfunktion darstellt. Aus diesem Grunde wird der bisher beste manuell bestimmte Parametersatz (siehe 5.1) als Maßstab verwendet. Findet der Optimierer im Erwartungswert vergleichbar gute Parametersätze, so zeigt dies, dass das implementierte Framework sich für die Bestimmung guter[1] Parameter eignet und die manuell zeitaufwändige Justierung durch das automatisiert laufende Framework ersetzt werden kann.

[1] gut vgl. zu dem manuell bestimmten Parametersatz

Parametersatz	Training	Validierung
$[-0,15; 1000; 1000; 5; 10; 0,001; 0,01]$	246,90 mm	193,94 mm

Tabelle 5.1: Bester manuell bestimmter Parametersatz mit der Durchschnittsabweichung bei Trainings- und Validierungsdatum

5.2 Konfiguration des PSO

Für alle Experimente wurde die modifizierte *Constriction PSO* wie in der nachfolgenden Tabelle 5.2 aufgeführt konfiguriert.

Bemerkung I Shi empfielt eine Populationsgröße von 20-50 (27). Um eine größere Streuung der Partikel zu erhalten, habe ich eine Schwarmgröße von 50 Partikel gewählt. Der festgelegte MaxStagnation-Wert ist explorativ gewählt worden. Ein zu niedriger Wert wie 2 führt dazu, dass die Partikel nicht die Möglichkeit haben sich in der neuen Position genug zu entfalten. Die Nachbarschaftsgröße ist auf 10 gesetzt, was beim Resampling 5 Nachbarschaften erstellt.

Bemerkung II Alle Versuche werden mehrmals durchgeführt, um festzustellen, wie robust der Optimierer funktioniert. Für die Durchführung wird das Trainingsdatum verwendet. Um das Ergebnis zu validieren, wird das

Parameter	Wert
Populationsgröße	50
Nachbarschaftsgröße	10
MaxGeneration	2000
MaxStagnation	10
Wertebereich *friction*	$[-0.5, -0.01]$
Wertebereich *process_update.invalid_error_x_threshold*	$[100, 1000]$
Wertebereich *process_update.invalid_error_y_threshold*	$[100, 1000]$
Wertebereich *process_update.noise_pos*	$[1e-08, 1000]$
Wertebereich *process_update.noise_vel*	$[1e-08, 1000]$
Wertebereich *sensor_update.noisepitch*	$[1e-08, 1000]$
Wertebereich *sensor_update.noiseyaw*	$[1e-08, 1000]$
Geschwindigkeit *friction*	$[-0.05, 0.05]$
Geschwindigkeit *process_update.invalid_error_x_threshold*	$[-200, 200]$
Geschwindigkeit *process_update.invalid_error_y_threshold*	$[-200, 200]$
Geschwindigkeit *process_update.noise_pos*	$[-200, 200]$
Geschwindigkeit *process_update.noise_vel*	$[-200, 200]$
Geschwindigkeit *sensor_update.noisepitch*	$[-200, 200]$
Geschwindigkeit *sensor_update.noiseyaw*	$[-200, 200]$

Tabelle 5.2: Konfiguration des *Constriction PSO* für alle Versuche

Validierungsdatum verwendet und jeweils der beste und schlechteste von allen Versuchen eines Experiments ausgewertet.

5.3 Experiment 1: Zufällige Initialisierung - Schnelles Ergebnis

5.3.1 Ziel und Beschreibung

Ziel Es gibt Szenarien, bei denen nur begrenzt viel Zeit für die Bestimmung von Parametersätzen zur Verfügung steht. Das folgende Experiment soll überprüfen, ob zur Referenz vergleichbare Parametersätze in wenigen Stunden durch den Optimierer bestimmbar sind.

Beschreibung Jedes Partikel der Population des PSO wird mit zufälligen, gleichverteilten Werten im zulässigen Wertebereich initialisiert. Um die schnelle Konvergenz der Partikel herbeizuführen, wurden als Lernfaktoren θ_1=3 und θ_2=5 gewählt, was das schnelle Zusammenziehen des Schwarms fördert. Das Experiment wurde 10 mal auf einem Rechner mit 6 CPU-Kernen durchgeführt und lief etwa 1-2 Stunden pro Durchlauf.

5.3.2 Ergebnisse und Evaluierung

Tabelle 5.3 zeigt die Ergebnisse des Trainings des 1. Experiments. Wie aus dieser hervorgeht, liegt die durchschnittliche Abweichung über alle 10 Versuche gemittelt bei 256,87 mm. Damit liegt die mittlere Abweichung mit dem automatisiert bestimmten Parametersatz im Durchschnitt 9,97 mm über der mittleren Abweichung des Referenzparametersatzes.

Durchlauf	θ_1=3, θ_2=5
1	256,20 mm
2	256,21 mm
3	256,46 mm
4	257,32 mm
5	257,14 mm
6	257,14 mm
7	256,18 mm
8	257,15 mm
9	257,16 mm
10	257,74 mm
Durchschnitt	256,87 mm

Tabelle 5.3: Übersicht der Ergebnisse aller Durchläufe von Experiment 1

$\theta_1 = 3, \theta_2 = 5$	Trainingsdatum	Validierungsdatum
schlechtester	257,74 mm	206,47 mm
bester	256,18 mm	203,07 mm
Referenz	246,90 mm	193,94 mm

Tabelle 5.4: Mittlere Abweichung der optimierten Parametersätze

Wie aus Tabelle 5.4 deutlich wird, ist auch beim Validierungsdatum eine geringe Differenz zwischen der mittleren Abweichung der automatisch bestimmten Parametersätze und der mittleren Abweichung des Referenzparametersatzes zu verzeichnen. Der Unterschied des besten Parametersatzes aller Durchläufe beträgt beim Trainingsdatum und beim Validierungsdatum 9,28 mm bzw. 9,13 mm über der mittleren Abweichung des Referenzparametersatzes. Im Falle des schlechtesten Parametersatzes liegt die Abweichung bei 10,84 mm bzw. 12,53 mm über der Referenz.

Evaluierung Das Experiment zeigt, dass vergleichbare Parametersätze in kurzer Zeit automatisiert bestimmt werden können. Bessere Parametersätze konnten bei diesem Experiment jedoch nicht gefunden werden. In allen Durchläufen betrug die Abweichung des global besten bereits nach einer Iteration < 300 mm. Diese "guten" Startwerte werfen jedoch die Frage auf, ob eine Verbesserung auch bei schlechten Startwerten erreicht werden kann. *Experiment 3* untersucht diese Fragestellung.

5.4 Experiment 2: Zufällige Initialisierung - Bessere Parameterbestimmung

5.4.1 Ziel und Beschreibung

Ziel Das Ziel dieses Experiments besteht darin zu zeigen, dass bei geeigneter Konfiguration und längerer Ausführungszeit bessere Parameter als die manuell bestimmten gefunden werden.

Beschreibung Jedes Partikel der Population des PSO wird mit zufälligen, gleichverteilten Werten im zulässigen Wertbereich initialisiert und der Optimierungsprozess gestartet. Optimiert wurde mit den Lernfaktoren θ_1=θ_2=2,05. Das Experiment wurde dreimalig durchgeführt. Die Ausführung erfolgte verteilt mit 6 Kernen und wurde 36 Stunden laufen lassen.

5.4.2 Ergebnisse und Evaluierung

Nr.	Trainingsdatum	Validierungsdatum
1	240,95 mm	188,94 mm
2	243,60 mm	190,14 mm
3	241,74 mm	188,92 mm
Durchschnitt	242,10 mm	189,33 mm

Tabelle 5.5: Mittlere Abweichung der optimierten Parametersätze

Nr.	Training	Valierung
1	29024 mm	28225 mm
2	25181 mm	28326 mm
3	16126 mm	21454 mm

Tabelle 5.6: Differenz der kumulierten Abweichungen

Ergebnisse Tabelle 5.5 zeigt die Ergebnisse der Optimierung. Aus ihr geht hervor, dass in allen drei Durchläufen ein besserer Parametersatz als die Referenz gefunden werden konnte. Dieses spiegelt sich auch im Validierungsdatum wieder. Die mittlere Abweichung des besten Parametersatzes liegt beim Trainingsdatum und Validierungsdatum 5,95 mm (Parametersatz 1) bzw. 5,02 mm (Parametersatz 3) niedriger im Vergleich zur mittleren Abweichung der Referenz. Beim schlechtesten unter den bestimmten Parametersätzen beträgt die Differenz zur Abweichung des manuell bestimmten 3,3 mm (Parametersatz 2) bzw. 3,8 mm (Parametersatz 2). Auffällig ist, dass trotz einer geringeren durchschnittlichen Abweichung von Parametersatz-Nr. 1 verglichen zu der durchschnittlichen Abweichung von Parametersatz 3 beim Trainingsdatum, die durchschnittliche Abweichung von Parametersatz 3 leicht geringer als die von Parametersatz 1 beim Validierungsdatum ist. Dieses deutet daraufhin, dass das Trainingsdatum zu kurz ist und weitere Beispiel-Frames benötigt werden. In Tabelle 5.6 ist die Differenz der aufsummierten Abweichung des Referenzparametersatzes zu den bestimmten Parametersätzen 1, 2 und 3 aufgeführt. Dieses zeigt, dass trotz der geringen Differenz in der mittleren Abweichung ein relativ hoher Betrag zustande kommt.
In der Abbildung 5.1 werden ausgewählte Partikelsamples illustriert, die sich anhand ihrer Form und Farbe unterscheiden. Das schwarze Kreuz markiert dabei den global besten nach Konvergierung des Optimierungsprozesses.

Evaluierung Das Experiment zeigt, dass unter Verwendung einer geeigneten Konfiguration und längerer Ausführung robust bessere Parametersätze bestimmbar sind, verglichen zu den manuell bestimmten.

5.5 Experiment 3: Optimierung schlechter Parametersätze

5.5.1 Ziel und Beschreibung

Ziel Das Ziel dieses Experiments ist es, zu zeigen, dass auch bei Vorliegen eines schlechten Samplings, robust eine Verbesserung durch das Verfahren erreicht werden kann.

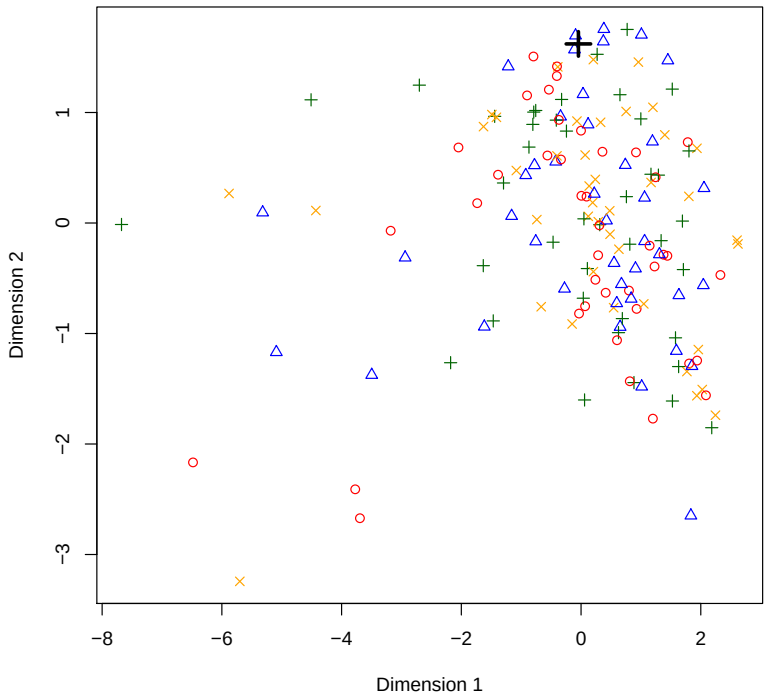

Abbildung 5.1: Zusammenziehen des Schwarms während der Optimierung, projiziert auf zwei Dimensionen mit Hilfe Multidimensionaler Skalierung

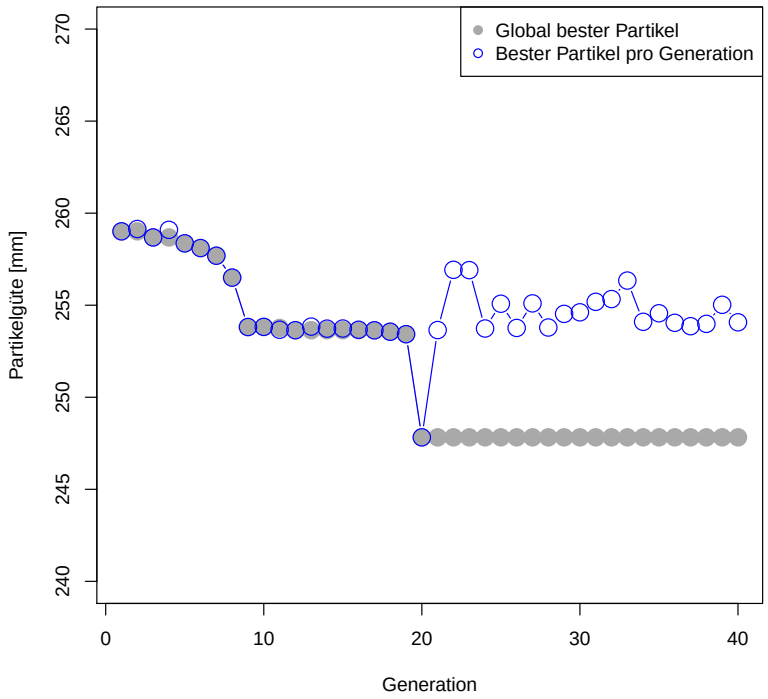

Abbildung 5.2: Visualisierung der Entwicklung der Abweichung des global besten bzw. des besten Partikel pro Generation während der ersten der drei Optimierungsdurchläufe

Beschreibung Drei beliebige, unterschiedliche Parametersätze mit einer schlechten Güte wurden bestimmt und der Optimierer nacheinander mit deren Werten initialisiert und gestartet. Die Optimierung wird pro Parametersatz 4-mal mit den Lernfaktoren $\theta_1 = 3$ und $\theta_2 = 5$ durchgeführt.
Tabelle 5.7 zeigt die drei ausgewählten Parametersätze. Ihre mittlere Abweichung für Trainings- und Validierungsdatum können Tabelle 5.8 entnommen werden.

Nr.	Parametersatz
1	$[-0,15; 1000,0; 1000,0; 1000,0; 722,0; 1e-08; 653,0]$
2	$[-0,15; 1000,0; 1000,0; 4,0; 0,5; 0,0006; 1e-08]$
3	$[-0,5; 989,0; 983,0; 960,0; 680,0; 1e-08; 619,0]$

Tabelle 5.7: Schlechte Parametersätze

Nr.	Trainingsdatum	Validierungsdatum
1	420,50 mm	292,43 mm
2	416,53 mm	270,31 mm
3	417,87 mm	289,26 mm

Tabelle 5.8: Mittlere Abweichung der schlechten Parametersätze

5.5.2 Ergebnisse und Evaluierung

Die nachfolgenden Tabellen (5.9- 5.11) zeigen die Ergebnisse der Optimierungsdurchläufe ausgehend von dem jeweiligen schlechten Parametersatz.

Durchlauf	Durchschnitt. Abweichung	Differenz	Verbesserung
1	258,35 mm	162,15 mm	38,56 %
2	258,34 mm	162,16 mm	38,56 %
3	258,33 mm	162,17 mm	38,57 %
4	258,36 mm	162,14 mm	38,56 %
Durchschnitt	258,35 mm	162,16 mm	38,56 %

Tabelle 5.9: Übersicht der Verbesserung von Parametersatz 1 mit ursprünglich 420,50 mm Abweichung im Durchschnitt

Durchlauf	Durchschnitt. Abweichung	Differenz	Verbesserung
1	259,36 mm	157,17 mm	37,73 %
2	253,32 mm	163,21 mm	39,18 %
3	242,45 mm	174,08 mm	41,79 %
4	253,46 mm	163,07 mm	39,15 %
Durchschnitt	252,15 mm	164,38 mm	39.46 %

Tabelle 5.10: Übersicht der Verbesserung von Parametersatz 2 mit ursprünglich 416,53 mm Abweichung im Durchschnitt

Durchlauf	Durchschnitt. Abweichung	Differenz	Verbesserung
1	258,40 mm	159,47 mm	38,16 %
2	258,40 mm	159,47 mm	38,16 %
3	258,32 mm	159,55 mm	38,18 %
4	258,32 mm	159,55 mm	38,18 %
Durchschnitt	258.36 mm	159,51 mm	38,08 %

Tabelle 5.11: Übersicht der Verbesserung von Parametersatz 3 mit ursprünglich 417,87 mm Abweichung im Durchschnitt

Die nachfolgende Tabelle zeigt die Validierung der besten und schlechtesten der drei Versuche.

Nr.	Bester	Verbesserung	Schlechtester	Verbesserung
1	206,41 mm	29,41 %	206,35 mm	29,43 %
2	189,88 mm	29,75 %	208,24 mm	22,96 %
3	206,33 mm	28,67 %	206,40 mm	28,65 %

Tabelle 5.12: Validierungsergebnisse der ermittelten Parameter

Evaluierung Das Experiment zeigt, dass die Optimierung ausgehend von schlechten Parametersätzen mit *Constriction PSO* und RSVN möglich ist. Die verwendeten Parameter wurden alle um etwa 37-42% beim Trainieren verbessert. Auch bei der Validierung konnte eine Verbesserung in Höhe von durchschnittlich 29,28% gemittelt über die besten aller Versuche und um durchschnittlich 27,91% gemittelt, über alle schlechtesten der Versuche, festgestellt werden. Auffällig, wenngleich nicht verwunderlich, ist der große Unterschied zwischen dem besten und dem schlechtesten des Versuches mit dem zweiten Parametersatz. Während zwischen dem besten und dem schlechtesten der beiden anderen Versuche lediglich eine Differenz von < 0,1 mm vorliegt, beträgt der Unterschied bei diesem > 18,26 mm.
Nach den vorherigen beiden Experimenten, welche zeigten, dass gute Parametersätze bestimmbar sind und schlechte sich verbessern lassen, ist eine weitere interessante Fragestellung, ob der manuell bestimmte Parametersatz durch den Optimierer verbessert werden kann. Das nächste Experiment evaluiert dieses.

5.6 Experiment 4: Optimierung des manuellen Parametersatzes

5.6.1 Ziel und Beschreibung

Ziel Mit Hilfe dieses Experiments soll überprüft werden, ob ausgehend von dem manuell bestimmten Parametersatz noch eine Verbesserung durch den Optimierer eintritt.

Beschreibung Alle Partikel der Population werden mit dem Referenzparametersatz initialisiert und der Optimierungsprozess gestartet. Der Optimierer wird mit den Lernfaktoren θ_1=1 und θ_2=5 bzw. $\theta_1 = \theta_2 = 2,05$ konfiguriert.

5.6.2 Ergebnisse und Evaluierung

Bemerkung Mit den Lernfaktoren θ_1=2,05 und θ_2=2,05 konnte in keinem der Durchläufe eine Verbesserung erzielt werden.

Tabelle 5.13 zeigt die mittlere Abweichung, die bei allen Durchläufen mit den Lernfaktoren θ_1=1 und θ_2=5 erreicht wurden. In allen Durchläufen konnte eine geringe Verbesserung erzielt werden. Die höchste Verbesserung beim Trainingsdatum beträgt 8,22 mm, was ca. 3,5 % entspricht; beim Validierungsdatum beträgt diese 2,88 mm (ca. 1,5 %).

In Tabelle 5.14 sind die mittlere Abweichung des besten und des schlechtesten unter den bestimmten Parametersätzen sowie der Referenz zusammengetragen.

Durchlauf	θ_1=1, θ_2=5
1	239,56 mm
2	239,42 mm
3	239,89 mm
4	239,07 mm
5	239,10 mm
6	238,81 mm
7	239,16 mm
8	239,16 mm
9	238,68 mm
10	239,35 mm
Durchschnitt	239,22 mm

Tabelle 5.13: Übersicht der Verbesserung

	Referenz	bester	schlechtester
Trainingsdatum	246,90 mm	238,68 mm	239,89 mm
Validierungsdatum	193,94 mm	191,06 mm	192,45 mm

Tabelle 5.14: Übersicht der durchschnittlichen Abweichungen

Evaluierung Wie aus Tabelle 5.14 deutlich wird, konnten mit den Lernfaktoren θ_1=1, θ_2=5 in allen Durchläufen eine Verbesserung der manuellen Parametersätze erreicht werden, wenngleich marginal. Eine wesentliche Verbesserung konnten nicht erzielt werden.

6
Schlussfolgerung

Zusammenfassung Partikelschwarmoptimierung wurde für die Bestimmung der Ballmodell-Parameter von humanoiden Fußball spielenden Robotern eingesetzt. Die ersten Gehversuche mit den Varianten RIW-, CIW- und LDIW-PSO unter Verwendung einer lbest-Topologie zeigten, dass diese bereits nach wenigen Iterationen für die zu optimierende Zielfunktion stagnieren. Die resultierten Parametersätze wichen dabei nur marginal von den bisher besten manuell bestimmten ab. Mit Hilfe einer modifizierten *constriction*-PSO Variante, die die Sampling-Strategie *Random Sampling in Variable Neighbourhoods* verwendet, um identifizierte Stagnationzustände zu behandeln, konnten bessere Ergebnisse erzielt werden. Mehrere Experimente mit dieser Variante wurden durchgeführt. Die Ergebnisse zeigen, dass sich das Verfahren für die Bestimmung der Ballmodell-Parameter eignet. Je nach Konfiguration des PSO lassen sich nach wenigen Iterationen bereits vergleichbare, wenngleich marginal schlechtere, Parametersätze verglichen zu den zeitaufwändig manuell justierten bestimmen. Eine längere Ausführung ermöglicht auch das Bestimmen besserer Parametersätze.

Parallelisierung Dank der Parallelisierung des Optimierungsframeworks konnte eine bis zu 30-fache Beschleunigung der Parameterauswertung erreicht werden. Die verteilte Berechnung wurde in einer eigens entwickelten Clusterumgebung mit bis zu 11 Rechnern und insgesamt 67 CPU-Kerne getestet. Es zeigte sich, das das Framework linear in der Anzahl der Kerne skaliert.

Diskussion Die Experimente haben gezeigt, dass das Verfahren sich für die Bestimmung von Parametersätzen für das Ballmodell eignet. Je nach Konfiguration lassen sich bereits nach wenigen Stunden vergleichbare Parametersätze und nach einer längeren Ausführung des Optimierers sogar bessere Parametersätze bestimmen als die bisher zeitaufwändig manuell justierten. Eine wesentliche Verbesserung blieb jedoch aus. Die Werte liegen trotz der Optimierung bei > 180 mm, was jedoch auf Probleme außerhalb der Computer Vision zurückgeführt werden dürfte. Eine der wesentlichen Ursachen für diese hohe Abweichung ist die niedrige Auflösung der Groundtruthkamera sowie der Blickwinkel auf das Spielfeld. Diese führt im Durchschnitt zu einer Abweichung von etwa 100 mm zwischen Groundtruthposition und der tatsächlichen Position des Balles. Ein weiteres Manko ist die Latenz der Groundtruthdaten, das bedeutet, die Groundtruthdaten kommen ver-

setzt an, so dass eine zeitliche Abweichung zwischen der Groundtruth und der geschätzten Ballposition vorliegt. Nichtzuletzt führen Probleme bei der Sensor-Fusion, der Synchronisation von Sensoren, zu Abweichungen.

Aussicht Um bessere Ergebnisse mit dem Verfahren zu erzielen, sollte im nächsten Schritt eine systematische Auseinandersetzung mit dem Einfluss der unterschiedlichen Kontrollparametern des PSO auf die vorliegende Zielfunktion erfolgen. Unter Verwendung einer besseren Kamera ließe sich die Genauigkeit der Groundtruthdaten erhöhen. Alternativ könnte auch die Auflösung der vorhandenen Kamera erhöht werden, was jedoch die Anzahl der Groundtruth-Frames verringert. Folglich müsste eine geeignete Strategie entwickelt werden, um zwischen den Frames zu interpolieren. Zu dem könnte untersucht werden, ob andere Sampling-Strategien und andere p-Normen sich besser eignen. Darüber hinaus sollte evaluiert werden, ob eine Verbesserung aus Seiten der Sensor-Fusion erreicht werden kann. Nichtzuletzt sollte das hier vorgestellte Framework verallgemeinert werden, um auch auf die Optimierung der Parameter anderer Modelle wie z.B. das Hindernismodell oder die Selbstlokalisierung angewendet werden zu können.

Literaturverzeichnis

[1] S. Otte, "Where am i? what's going on? – world modelling using multi-hypothesis kalman filters for humanoid soccer robots," master thesis, Freie Universität Berlin, December 2012.

[2] http://www.robocup.org/about-robocup/objective/, [10.02.2014].

[3] R. K. Jatoth and T. K. Kumar, "Particle swarm optimization based tuning of extended kalman filter for manoeuvring target tracking," in *Mathematics and Computers In Science And Engineering archive Proceedings of the 8th WSEAS international conference on Signal processing, robotics and automation, Cambridge 2009*, 2009.

[4] X. Yang, A. Gandomi, S. Talatahari, and A. Alavi, *Metaheuristics in Water, Geotechnical and Transport Engineering.* Elsevier insights, Elsevier Science, 2012.

[5] B. Majhi and G. Panda, "Distributed and robust parameter estimation of iir systems using incremental particle swarm optimization.," *Digital Signal Processing*, vol. 23, no. 4, pp. 1303–1313, 2013.

[6] J. Wang, S. Zhu, W. Zhao, and W. Zhu, "Optimal parameters estimation and input subset for grey model based on chaotic particle swarm optimization algorithm.," *Expert Syst. Appl.*, vol. 38, no. 7, pp. 8151–8158, 2011.

[7] J. Zhou, R. Fang, Y. Li, Y. Zhang, and B. Peng, "Parameter optimization of nonlinear grey bernoulli model using particle swarm optimization.," *Applied Mathematics and Computation*, vol. 207, no. 2, pp. 292–299, 2009.

[8] C. Niehaus, T. Röfer, and T. Laue, "Gait optimization on a humanoid robot using particle swarm optimization," in *Proceedings of the Second Workshop on Humanoid Soccer Robots in conjunction with the 2007 IEEE-RAS International Conference on Humanoid Robots* (C. Zhou, E. Pagello, E. Menegatti, and S. Behnke, eds.), o.A., 2007.

[9] A. Burchardt, T. Laue, and T. Röfer, "Optimizing particle filter parameters for self-localization.," in *RobuCup* (J. R. del Solar, E. Chown, and P.-G. Plöger, eds.), vol. 6556 of *Lecture Notes in Computer Science*, pp. 145–156, Springer, 2010.

[10] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems.* Santa Fe Institute Studies in the Sciences of Complexity Series, Oxford Press, July 1999.

[11] D. Simon, *Evolutionary Optimization Algorithms*. Wiley, 2013.

[12] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Tech. Rep. TR06, Erciyes University, Oct. 2005.

[13] O. Brodersen and S. M., "Einsatz der particle swarm optimization zur optimierung universitärer stundenpläne," tech. rep., Georg-August-Universität Göttingen, 2007.

[14] M. Karatay. http://commons.wikimedia.org/wiki/File:Safari_ants.jpg, [10.02.2014], 05 2007.

[15] B. Inaglory. http://en.wikipedia.org/wiki/File:School_of_Pterocaesio_chrysozona_in_Papua_New_Guinea_1.jpg, [10.02.2014], 04 2004.

[16] G. Beni, "From swarm intelligence to swarm robotics.," in *Swarm Robotics* (E. Sahin and W. M. Spears, eds.), vol. 3342 of *Lecture Notes in Computer Science*, pp. 1–9, Springer, 2004.

[17] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, April 1997.

[18] A. Abdelbar and S. Abdelshahid, "Instinct-based pso with local search applied to satisfiability," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, vol. 3, pp. 2291–2295 vol.3, 2004.

[19] J. Sun, J. Liu, and W. Xu, "Using quantum-behaved particle swarm optimization algorithm to solve non-linear programming problems.," *Int. J. Comput. Math.*, vol. 84, no. 2, pp. 261–272, 2007.

[20] I. Kassabalidis, M. El-Sharkawi, R. J. M. II, P. Arabshahi, and A. A. Gray, "Swarm intelligence for routing in communication networks," in *Proc. IEEE GlobeComm*, November 2001.

[21] X. Zhang, X. Chen, and Z. He, "An aco-based algorithm for parameter optimization of support vector machines.," *Expert Syst. Appl.*, vol. 37, no. 9, pp. 6618–6628, 2010.

[22] M. Rosendo and A. Pozo, "Applying a discrete particle swarm optimization algorithm to combinatorial problems.," in *SBRN* (T. B. Ludermir, K. Figueiredo, and C. E. Thomaz, eds.), pp. 235–240, IEEE, 2010.

[23] R. C. Eberhart and J. Kennedy, "Particle swarm optimization," in *Particle swarm optimization*, vol. 4, pp. 1942–1948, 1995.

[24] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," 1987.

[25] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks.," *American Association for the Advancement of science, Washington, DC(USA). 1990.*, 1990.

[26] U. Paquet and A. P. Engelbrecht, "A new particle swarm optimiser for linearly constrained optimisation.," in *IEEE Congress on Evolutionary Computation (1)*, pp. 227–233, IEEE, 2003.

[27] Y. Shi *et al.*, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, pp. 81–86, IEEE, 2001.

[28] Y. Shi and R. C. Eberhart, "A Modified Particle Swarm Optimizer," in *Proceedings of IEEE International Conference on Evolutionary Computation*, (Washington, DC, USA), pp. 69–73, IEEE Computer Society, May 1998.

[29] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham, "Inertia weight strategies in particle swarm optimization.," in *NaBIC*, pp. 633–640, IEEE, 2011.

[30] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European Journal of Operational Research*, vol. In Press, Corrected Proof, 2006.

[31] T. Liao, D. Molina, T. Stützle, M. A. M. de Oca, and M. Dorigo, "An aco algorithm benchmarked on the bbob noiseless function testbed.," in *GECCO (Companion)* (T. Soule and J. H. Moore, eds.), pp. 159–166, ACM, 2012.

[32] G. Nápoles, I. Grau, and R. Bello, "Constricted particle swarm optimization based algorithm for global optimization," 2012.